



THIRD EDITION

Use with
Java™ SE 7
or Java™ SE 8

Java™ SE 8 for Programmers

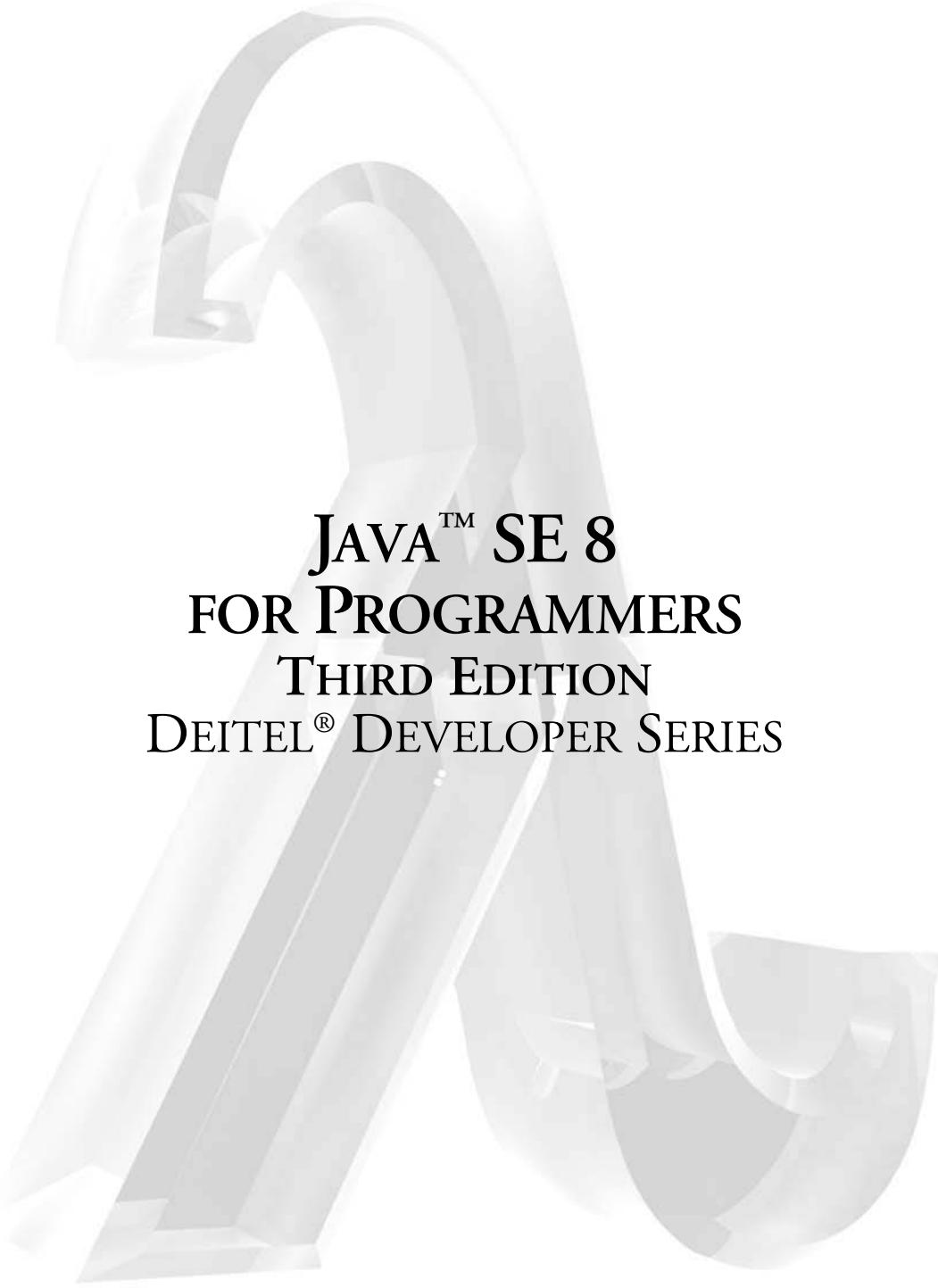


PAUL DEITEL • HARVEY DEITEL

FREE SAMPLE CHAPTER

SHARE WITH OTHERS





JAVA™ SE 8
FOR PROGRAMMERS
THIRD EDITION
DEITEL® DEVELOPER SERIES

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors 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 U.S., please contact international@pearsoned.com.

Visit us on the web: informit.com/ph

Library of Congress Cataloging-in-Publication Data

On file

© 2014 Pearson Education, Inc.

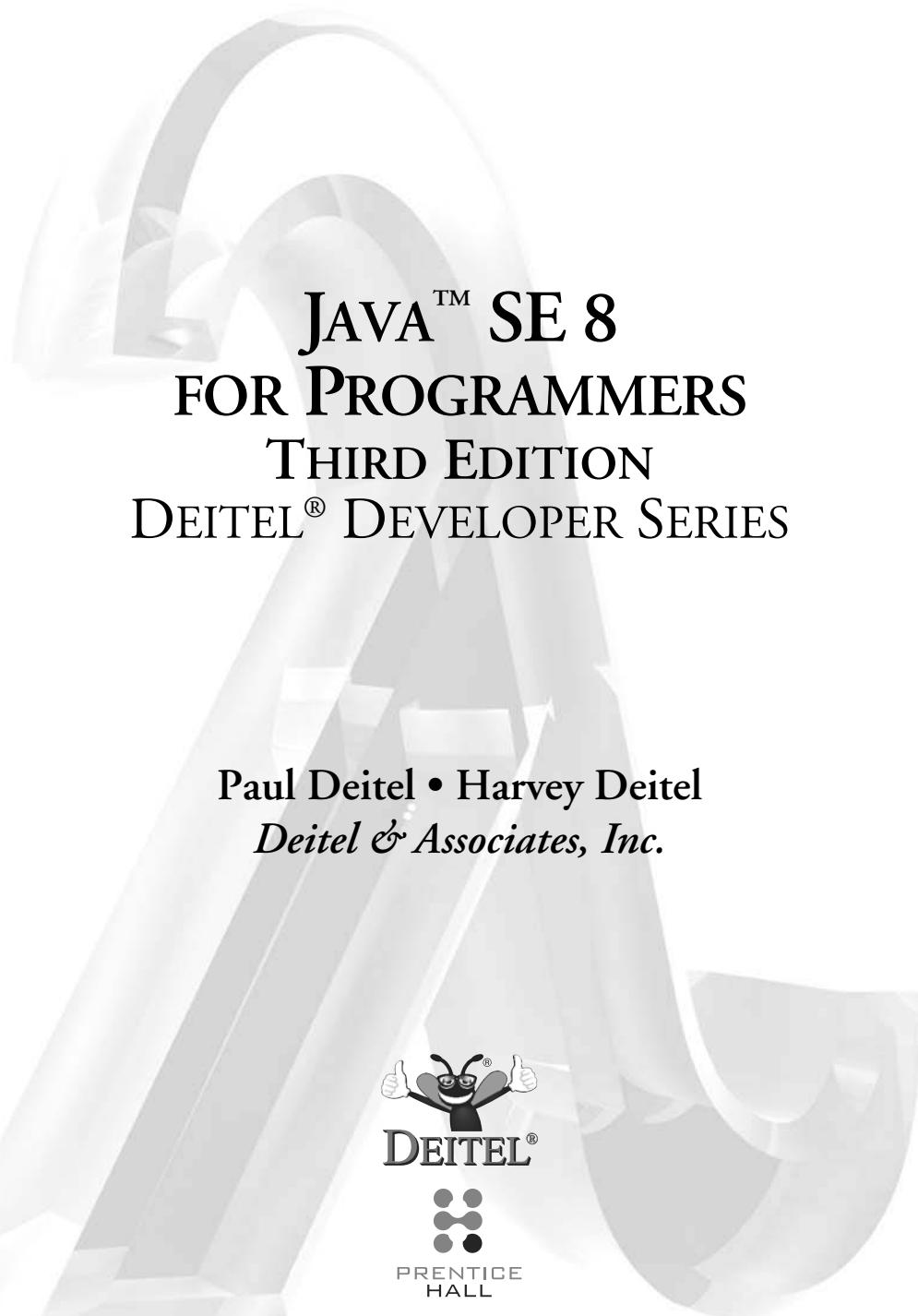
Portions of the cover are modifications based on work created and shared by Google (<http://code.google.com/policies.html>) and used according to terms described in the Creative Commons 3.0 Attribution License (<http://creativecommons.org/licenses/by/3.0/>).

All rights reserved. Printed in the United States of America. 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. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-13389138-6

ISBN-10: 0-13-389138-0

Text printed in the United States at Edwards Brothers Malloy in Ann Arbor, Michigan.
First printing, March 2014



JAVA™ SE 8 FOR PROGRAMMERS

THIRD EDITION

DEITEL® DEVELOPER SERIES

Paul Deitel • Harvey Deitel
Deitel & Associates, Inc.



Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Deitel® Series Page

Deitel® Developer Series

Android for Programmers: An App-Driven Approach, 2/E, Volume 1
C for Programmers with an Introduction to C11
C++11 for Programmers
C# 2012 for Programmers
Dive Into® iOS 6 for Programmers: An App-Driven Approach
Java™ for Programmers, 3/E
JavaScript for Programmers

How To Program Series

Android How to Program, 2/E
C++ How to Program, 9/E
C How to Program, 7/E
Java™ How to Program, 10/E
Java™ How to Program, Late Objects Version, 10/E
Internet & World Wide Web How to Program, 5/E
Visual C++® 2008 How to Program, 2/E
Visual Basic® 2012 How to Program, 6/E
Visual C#® 2012 How to Program, 5/E

Simply Series

Simply C++: An App-Driven Tutorial Approach
Simply Java™ Programming: An App-Driven Tutorial Approach

(continued from previous column)

Simply C#: An App-Driven Tutorial Approach
Simply Visual Basic® 2010: An App-Driven Approach, 4/E

CourseSmart Web Books

www.deitel.com/books/CourseSmart/
C++ How to Program, 8/E and 9/E
Simply C++: An App-Driven Tutorial Approach
Java™ How to Program, 9/E and 10/E
Simply Visual Basic® 2010: An App-Driven Approach, 4/E
Visual Basic® 2012 How to Program, 6/E
Visual Basic® 2010 How to Program, 5/E
Visual C#® 2012 How to Program, 5/E
Visual C#® 2010 How to Program, 4/E

LiveLessons Video Learning Products

www.deitel.com/books/LiveLessons/
Android App Development Fundamentals
C++ Fundamentals
Java™ Fundamentals
C# 2012 Fundamentals
C# 2010 Fundamentals
iOS® 6 App Development Fundamentals
JavaScript Fundamentals
Visual Basic® Fundamentals

To receive updates on Deitel publications, Resource Centers, training courses, partner offers and more, please join the Deitel communities on

- Facebook®—facebook.com/DeitelFan
- Twitter®—@deitel
- Google+™—google.com/+DeitelFan
- YouTube™—youtube.com/DeitelTV
- LinkedIn®—linkedin.com/company/deitel-&-associates

and register for the free *Deitel® Buzz Online* e-mail newsletter at:

www.deitel.com/newsletter/subscribe.html

To communicate with the authors, send e-mail to:

deitel@deitel.com

For information on *Dive-Into® Series* on-site seminars offered by Deitel & Associates, Inc. worldwide, write to us at deitel@deitel.com or visit:

www.deitel.com/training/

For continuing updates on Pearson/Deitel publications visit:

www.deitel.com

www.pearsonhighered.com/deitel/

Visit the Deitel Resource Centers that will help you master programming languages, software development, Android and iOS app development, and Internet- and web-related topics:

www.deitel.com/ResourceCenters.html

*To Brian Goetz,
Oracle's Java Language Architect and
Specification Lead for Java SE 8's Project Lambda:*

*Your mentorship helped us make a better book.
Thank you for insisting that we get it right.*

Paul and Harvey Deitel

Trademarks

DEITEL, the double-thumbs-up bug and DIVE-INTO are registered trademarks of Deitel & Associates, Inc.

Java is a registered trademark of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Google, Android, Google Play, Google Maps, Google Wallet, Nexus, YouTube, AdSense and AdMob are trademarks of Google, Inc.

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® and Windows® are registered trademarks of the Microsoft Corporation in the U.S.A. and other countries. Screen shots and icons reprinted with permission from the Microsoft Corporation. This book is not sponsored or endorsed by or affiliated with the Microsoft Corporation.

Throughout this book, trademarks are used. Rather than put a trademark symbol in every occurrence of a trademarked name, we state that we are using the names in an editorial fashion only and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Contents

Foreword	xxiii
Preface	xxv
Before You Begin	xxxvii
I Introduction to Java and Test-Driving a Java Application	I
1.1 Introduction	2
1.2 Object Technology Concepts	4
1.2.1 The Automobile as an Object	4
1.2.2 Methods and Classes	4
1.2.3 Instantiation	4
1.2.4 Reuse	5
1.2.5 Messages and Method Calls	5
1.2.6 Attributes and Instance Variables	5
1.2.7 Encapsulation and Information Hiding	5
1.2.8 Inheritance	5
1.2.9 Interfaces	6
1.2.10 Object-Oriented Analysis and Design (OOAD)	6
1.2.11 The UML (Unified Modeling Language)	6
1.3 Open Source Software	7
1.4 Java	8
1.5 A Typical Java Development Environment	9
1.6 Test-Driving a Java Application	12
1.7 Software Technologies	16
1.8 Keeping Up-to-Date with Information Technologies	18
2 Introduction to Java Applications; Input/Output and Operators	20
2.1 Introduction	21
2.2 Your First Program in Java: Printing a Line of Text	21
2.3 Modifying Your First Java Program	26
2.4 Displaying Text with <code>printf</code>	28

2.5	Another Application: Adding Integers	29
2.6	Arithmetic	33
2.7	Decision Making: Equality and Relational Operators	34
2.8	Wrap-Up	37

3 Introduction to Classes, Objects, Methods and Strings 38

3.1	Introduction	39
3.2	Instance Variables, <i>set</i> Methods and <i>get</i> Methods	39
3.2.1	Account Class with an Instance Variable, a <i>set</i> Method and a <i>get</i> Method	40
3.2.2	AccountTest Class That Creates and Uses an Object of Class Account	43
3.2.3	Compiling and Executing an App with Multiple Classes	46
3.2.4	Account UML Class Diagram with an Instance Variable and <i>set</i> and <i>get</i> Methods	46
3.2.5	Additional Notes on Class AccountTest	47
3.2.6	Software Engineering with private Instance Variables and public <i>set</i> and <i>get</i> Methods	48
3.3	Primitive Types vs. Reference Types	49
3.4	Account Class: Initializing Objects with Constructors	50
3.4.1	Declaring an Account Constructor for Custom Object Initialization	50
3.4.2	Class AccountTest: Initializing Account Objects When They're Created	51
3.5	Account Class with a Balance; Floating-Point Numbers	53
3.5.1	Account Class with a <i>balance</i> Instance Variable of Type double	54
3.5.2	AccountTest Class to Use Class Account	55
3.6	Wrap-Up	58

4 Control Statements: Part I; Assignment, **++** and **--** Operators 59

4.1	Introduction	60
4.2	Control Structures	60
4.3	if Single-Selection Statement	62
4.4	if...else Double-Selection Statement	63
4.5	Student Class: Nested if...else Statements	67
4.6	while Repetition Statement	69
4.7	Counter-Controlled Repetition	71
4.8	Sentinel-Controlled Repetition	74
4.9	Nested Control Statements	79
4.10	Compound Assignment Operators	81
4.11	Increment and Decrement Operators	81
4.12	Primitive Types	84
4.13	Wrap-Up	85

5	Control Statements: Part 2; Logical Operators	86
5.1	Introduction	87
5.2	Essentials of Counter-Controlled Repetition	87
5.3	for Repetition Statement	88
5.4	Examples Using the for Statement	92
5.5	do...while Repetition Statement	97
5.6	switch Multiple-Selection Statement	98
5.7	Class AutoPolicy Case Study: Strings in switch Statements	104
5.8	break and continue Statements	108
5.9	Logical Operators	110
5.10	Wrap-Up	115
6	Methods: A Deeper Look	117
6.1	Introduction	118
6.2	Program Modules in Java	118
6.3	static Methods, static Fields and Class Math	119
6.4	Declaring Methods with Multiple Parameters	121
6.5	Notes on Declaring and Using Methods	124
6.6	Argument Promotion and Casting	125
6.7	Java API Packages	127
6.8	Case Study: Secure Random-Number Generation	128
6.9	Case Study: A Game of Chance; Introducing enum Types	133
6.10	Scope of Declarations	138
6.11	Method Overloading	140
6.12	Wrap-Up	142
7	Arrays and ArrayLists	144
7.1	Introduction	145
7.2	Arrays	146
7.3	Declaring and Creating Arrays	147
7.4	Examples Using Arrays	148
7.4.1	Creating and Initializing an Array	148
7.4.2	Using an Array Initializer	149
7.4.3	Calculating the Values to Store in an Array	150
7.4.4	Summing the Elements of an Array	152
7.4.5	Using Bar Charts to Display Array Data Graphically	152
7.4.6	Using the Elements of an Array as Counters	154
7.4.7	Using Arrays to Analyze Survey Results	155
7.5	Exception Handling: Processing the Incorrect Response	157
7.5.1	The try Statement	157
7.5.2	Executing the catch Block	157
7.5.3	toString Method of the Exception Parameter	158
7.6	Case Study: Card Shuffling and Dealing Simulation	158
7.7	Enhanced for Statement	163

7.8	Passing Arrays to Methods	164
7.9	Pass-By-Value vs. Pass-By-Reference	166
7.10	Case Study: Class GradeBook Using an Array to Store Grades	167
7.11	Multidimensional Arrays	173
7.12	Case Study: Class GradeBook Using a Two-Dimensional Array	176
7.13	Variable-Length Argument Lists	182
7.14	Using Command-Line Arguments	184
7.15	Class Arrays	186
7.16	Introduction to Collections and Class ArrayList	188
7.17	Wrap-Up	192

8 Classes and Objects: A Deeper Look 193

8.1	Introduction	194
8.2	Time Class Case Study	194
8.3	Controlling Access to Members	199
8.4	Referring to the Current Object's Members with the this Reference	200
8.5	Time Class Case Study: Overloaded Constructors	202
8.6	Default and No-Argument Constructors	208
8.7	Notes on Set and Get Methods	208
8.8	Composition	210
8.9	enum Types	213
8.10	Garbage Collection	215
8.11	static Class Members	216
8.12	static Import	220
8.13	final Instance Variables	221
8.14	Time Class Case Study: Creating Packages	222
8.15	Package Access	228
8.16	Using BigDecimal for Precise Monetary Calculations	230
8.17	Wrap-Up	232

9 Object-Oriented Programming: Inheritance 234

9.1	Introduction	235
9.2	Superclasses and Subclasses	236
9.3	protected Members	238
9.4	Relationship Between Superclasses and Subclasses	239
9.4.1	Creating and Using a CommissionEmployee Class	239
9.4.2	Creating and Using a BasePlusCommissionEmployee Class	245
9.4.3	Creating a CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy	250
9.4.4	CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using protected Instance Variables	253
9.4.5	CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using private Instance Variables	256
9.5	Constructors in Subclasses	261

9.6	Class Object	261
9.7	Wrap-Up	262

10 Object-Oriented Programming: Polymorphism and Interfaces **264**

10.1	Introduction	265
10.2	Polymorphism Examples	267
10.3	Demonstrating Polymorphic Behavior	268
10.4	Abstract Classes and Methods	270
10.5	Case Study: Payroll System Using Polymorphism	273
10.5.1	Abstract Superclass Employee	274
10.5.2	Concrete Subclass SalariedEmployee	276
10.5.3	Concrete Subclass HourlyEmployee	278
10.5.4	Concrete Subclass CommissionEmployee	280
10.5.5	Indirect Concrete Subclass BasePlusCommissionEmployee	282
10.5.6	Polymorphic Processing, Operator instanceof and Downcasting	283
10.6	Allowed Assignments Between Superclass and Subclass Variables	288
10.7	final Methods and Classes	288
10.8	A Deeper Explanation of Issues with Calling Methods from Constructors	289
10.9	Creating and Using Interfaces	290
10.9.1	Developing a Payable Hierarchy	291
10.9.2	Interface Payable	292
10.9.3	Class Invoice	293
10.9.4	Modifying Class Employee to Implement Interface Payable	295
10.9.5	Modifying Class SalariedEmployee for Use in the Payable Hierarchy	297
10.9.6	Using Interface Payable to Process Invoices and Employees Polymorphically	299
10.9.7	Some Common Interfaces of the Java API	300
10.10	Java SE 8 Interface Enhancements	301
10.10.1	default Interface Methods	301
10.10.2	static Interface Methods	302
10.10.3	Functional Interfaces	302
10.11	Wrap-Up	302

11 Exception Handling: A Deeper Look **304**

11.1	Introduction	305
11.2	Example: Divide by Zero without Exception Handling	306
11.3	Exception Handling: ArithmeticExceptions and InputMismatchExceptions	308
11.4	When to Use Exception Handling	314
11.5	Java Exception Hierarchy	314
11.6	finally Block	317
11.7	Stack Unwinding and Obtaining Information from an Exception Object	322

11.8	Chained Exceptions	324
11.9	Declaring New Exception Types	327
11.10	Preconditions and Postconditions	327
11.11	Assertions	328
11.12	try-with-Resources: Automatic Resource Deallocation	330
11.13	Wrap-Up	330

12 Swing GUI Components: Part I 332

12.1	Introduction	333
12.2	Java's Nimbus Look-and-Feel	334
12.3	Simple GUI-Based Input/Output with JOptionPane	335
12.4	Overview of Swing Components	338
12.5	Displaying Text and Images in a Window	340
12.6	Text Fields and an Introduction to Event Handling with Nested Classes	344
12.7	Common GUI Event Types and Listener Interfaces	350
12.8	How Event Handling Works	352
12.9	JButton	354
12.10	Buttons That Maintain State	357
12.10.1	JCheckBox	358
12.10.2	JRadioButton	360
12.11	JComboBox; Using an Anonymous Inner Class for Event Handling	363
12.12	JList	367
12.13	Multiple-Selection Lists	370
12.14	Mouse Event Handling	372
12.15	Adapter Classes	377
12.16	JPanel Subclass for Drawing with the Mouse	381
12.17	Key Event Handling	384
12.18	Introduction to Layout Managers	387
12.18.1	FlowLayout	389
12.18.2	BorderLayout	391
12.18.3	GridLayout	395
12.19	Using Panels to Manage More Complex Layouts	397
12.20	JTextArea	398
12.21	Wrap-Up	401

13 Graphics and Java 2D 402

13.1	Introduction	403
13.2	Graphics Contexts and Graphics Objects	405
13.3	Color Control	406
13.4	Manipulating Fonts	413
13.5	Drawing Lines, Rectangles and Ovals	418
13.6	Drawing Arcs	422
13.7	Drawing Polygons and Polylines	425
13.8	Java 2D API	428
13.9	Wrap-Up	435

14 Strings, Characters and Regular Expressions	436
14.1 Introduction	437
14.2 Fundamentals of Characters and Strings	437
14.3 Class String	438
14.3.1 String Constructors	438
14.3.2 String Methods length, charAt and getChars	439
14.3.3 Comparing Strings	440
14.3.4 Locating Characters and Substrings in Strings	445
14.3.5 Extracting Substrings from Strings	447
14.3.6 Concatenating Strings	448
14.3.7 Miscellaneous String Methods	448
14.3.8 String Method valueOf	450
14.4 Class StringBuilder	451
14.4.1 StringBuilder Constructors	452
14.4.2 StringBuilder Methods length, capacity, setLength and ensureCapacity	452
14.4.3 StringBuilder Methods charAt, setCharAt, getChars and reverse	454
14.4.4 StringBuilder append Methods	455
14.4.5 StringBuilder Insertion and Deletion Methods	457
14.5 Class Character	458
14.6 Tokenizing Strings	463
14.7 Regular Expressions, Class Pattern and Class Matcher	464
14.8 Wrap-Up	473
15 Files, Streams and Object Serialization	474
15.1 Introduction	475
15.2 Files and Streams	475
15.3 Using NIO Classes and Interfaces to Get File and Directory Information	477
15.4 Sequential-Access Text Files	481
15.4.1 Creating a Sequential-Access Text File	481
15.4.2 Reading Data from a Sequential-Access Text File	485
15.4.3 Case Study: A Credit-Inquiry Program	487
15.4.4 Updating Sequential-Access Files	491
15.5 Object Serialization	492
15.5.1 Creating a Sequential-Access File Using Object Serialization	493
15.5.2 Reading and Deserializing Data from a Sequential-Access File	498
15.6 Opening Files with JFileChooser	500
15.7 (Optional) Additional java.io Classes	503
15.7.1 Interfaces and Classes for Byte-Based Input and Output	503
15.7.2 Interfaces and Classes for Character-Based Input and Output	505
15.8 Wrap-Up	506

16	Generic Collections	507
16.1	Introduction	508
16.2	Collections Overview	508
16.3	Type-Wrapper Classes	510
16.4	Autoboxing and Auto-Unboxing	510
16.5	Interface Collection and Class Collections	510
16.6	Lists	511
16.6.1	ArrayList and Iterator	512
16.6.2	LinkedList	514
16.7	Collections Methods	519
16.7.1	Method sort	520
16.7.2	Method shuffle	523
16.7.3	Methods reverse, fill, copy, max and min	525
16.7.4	Method binarySearch	527
16.7.5	Methods addAll, frequency and disjoint	529
16.8	Stack Class of Package java.util	531
16.9	Class PriorityQueue and Interface Queue	533
16.10	Sets	534
16.11	Maps	537
16.12	Properties Class	541
16.13	Synchronized Collections	544
16.14	Unmodifiable Collections	544
16.15	Abstract Implementations	545
16.16	Wrap-Up	545
17	Java SE 8 Lambdas and Streams	547
17.1	Introduction	548
17.2	Functional Programming Technologies Overview	549
17.2.1	Functional Interfaces	550
17.2.2	Lambda Expressions	551
17.2.3	Streams	552
17.3	IntStream Operations	554
17.3.1	Creating an IntStream and Displaying Its Values with the forEach Terminal Operation	556
17.3.2	Terminal Operations count, min, max, sum and average	557
17.3.3	Terminal Operation reduce	557
17.3.4	Intermediate Operations: Filtering and Sorting IntStream Values	559
17.3.5	Intermediate Operation: Mapping	560
17.3.6	Creating Streams of ints with IntStream Methods range and rangeClosed	561
17.4	Stream<Integer> Manipulations	561
17.4.1	Creating a Stream<Integer>	562
17.4.2	Sorting a Stream and Collecting the Results	563
17.4.3	Filtering a Stream and Storing the Results for Later Use	563

17.4.4	Filtering and Sorting a Stream and Collecting the Results	563
17.4.5	Sorting Previously Collected Results	563
17.5	Stream<String> Manipulations	564
17.5.1	Mapping Strings to Uppercase Using a Method Reference	565
17.5.2	Filtering Strings Then Sorting Them in Case-Insensitive Ascending Order	566
17.5.3	Filtering Strings Then Sorting Them in Case-Insensitive Descending Order	566
17.6	Stream<Employee> Manipulations	566
17.6.1	Creating and Displaying a List<Employee>	568
17.6.2	Filtering Employees with Salaries in a Specified Range	569
17.6.3	Sorting Employees By Multiple Fields	570
17.6.4	Mapping Employees to Unique Last Name Strings	572
17.6.5	Grouping Employees By Department	573
17.6.6	Counting the Number of Employees in Each Department	574
17.6.7	Summing and Averaging Employee Salaries	574
17.7	Creating a Stream<String> from a File	576
17.8	Generating Streams of Random Values	579
17.9	Lambda Event Handlers	581
17.10	Additional Notes on Java SE 8 Interfaces	581
17.11	Java SE 8 and Functional Programming Resources	582
17.12	Wrap-Up	582

18 Generic Classes and Methods 584

18.1	Introduction	585
18.2	Motivation for Generic Methods	585
18.3	Generic Methods: Implementation and Compile-Time Translation	587
18.4	Additional Compile-Time Translation Issues: Methods That Use a Type Parameter as the Return Type	590
18.5	Overloading Generic Methods	593
18.6	Generic Classes	594
18.7	Raw Types	601
18.8	Wildcards in Methods That Accept Type Parameters	605
18.9	Wrap-Up	609

19 Swing GUI Components: Part 2 611

19.1	Introduction	612
19.2	JSlider	612
19.3	Understanding Windows in Java	616
19.4	Using Menus with Frames	617
19.5	JPopupMenu	625
19.6	Pluggable Look-and-Feel	628
19.7	JDesktopPane and JInternalFrame	633
19.8	JTabbedPane	636

19.9	BoxLayout Layout Manager	638
19.10	GridBagLayout Layout Manager	642
19.11	Wrap-Up	652

20 Concurrency 653

20.1	Introduction	654
20.2	Thread States and Life Cycle	656
20.2.1	<i>New</i> and <i>Runnable</i> States	657
20.2.2	<i>Waiting</i> State	657
20.2.3	<i>Timed Waiting</i> State	657
20.2.4	<i>Blocked</i> State	657
20.2.5	<i>Terminated</i> State	657
20.2.6	Operating-System View of the <i>Runnable</i> State	658
20.2.7	Thread Priorities and Thread Scheduling	658
20.2.8	Indefinite Postponement and Deadlock	659
20.3	Creating and Executing Threads with the Executor Framework	659
20.4	Thread Synchronization	663
20.4.1	Immutable Data	664
20.4.2	Monitors	664
20.4.3	Unsynchronized Mutable Data Sharing	665
20.4.4	Synchronized Mutable Data Sharing—Making Operations Atomic	670
20.5	Producer/Consumer Relationship without Synchronization	672
20.6	Producer/Consumer Relationship: <code>ArrayBlockingQueue</code>	680
20.7	(Advanced) Producer/Consumer Relationship with <code>synchronized</code> , <code>wait</code> , <code>notify</code> and <code>notifyAll</code>	683
20.8	(Advanced) Producer/Consumer Relationship: Bounded Buffers	690
20.9	(Advanced) Producer/Consumer Relationship: The Lock and Condition Interfaces	698
20.10	Concurrent Collections	705
20.11	Multithreading with GUI: <code>SwingWorker</code>	707
20.11.1	Performing Computations in a Worker Thread: Fibonacci Numbers	708
20.11.2	Processing Intermediate Results: Sieve of Eratosthenes	714
20.12	<code>sort/parallelSort</code> Timings with the Java SE 8 Date/Time API	721
20.13	Java SE 8: Sequential vs. Parallel Streams	723
20.14	(Advanced) Interfaces <code>Callable</code> and <code>Future</code>	726
20.15	(Advanced) Fork/Join Framework	730
20.16	Wrap-Up	730

21 Accessing Databases with JDBC 732

21.1	Introduction	733
21.2	Relational Databases	734
21.3	A books Database	735
21.4	SQL	739

21.4.1	Basic SELECT Query	739
21.4.2	WHERE Clause	740
21.4.3	ORDER BY Clause	742
21.4.4	Merging Data from Multiple Tables: INNER JOIN	743
21.4.5	INSERT Statement	745
21.4.6	UPDATE Statement	746
21.4.7	DELETE Statement	747
21.5	Setting up a Java DB Database	747
21.5.1	Creating the Chapter’s Databases on Windows	748
21.5.2	Creating the Chapter’s Databases on Mac OS X	749
21.5.3	Creating the Chapter’s Databases on Linux	750
21.6	Manipulating Databases with JDBC	750
21.6.1	Connecting to and Querying a Database	750
21.6.2	Querying the books Database	754
21.7	RowSet Interface	767
21.8	PreparedStatements	769
21.9	Stored Procedures	785
21.10	Transaction Processing	785
21.11	Wrap-Up	786

22 JavaFX GUI

787

22.1	Introduction	788
22.2	JavaFX Scene Builder and the NetBeans IDE	789
22.3	JavaFX App Window Structure	790
22.4	Welcome App—Displaying Text and an Image	791
22.4.1	Creating the App’s Project	791
22.4.2	NetBeans Projects Window—Viewing the Project Contents	793
22.4.3	Adding an Image to the Project	794
22.4.4	Opening JavaFX Scene Builder from NetBeans	794
22.4.5	Changing to a VBox Layout Container	795
22.4.6	Configuring the VBox Layout Container	796
22.4.7	Adding and Configuring a Label	796
22.4.8	Adding and Configuring an ImageView	796
22.4.9	Running the Welcome App	797
22.5	Tip Calculator App—Introduction to Event Handling	798
22.5.1	Test-Driving the Tip Calculator App	799
22.5.2	Technologies Overview	799
22.5.3	Building the App’s GUI	802
22.5.4	TipCalculator Class	806
22.5.5	TipCalculatorController Class	808
22.6	Wrap-Up	813

23 ATM Case Study, Part I: Object-Oriented Design with the UML

815

23.1	Case Study Introduction	816
------	-------------------------	-----

23.2	Examining the Requirements Document	816
23.3	Identifying the Classes in a Requirements Document	824
23.4	Identifying Class Attributes	830
23.5	Identifying Objects' States and Activities	835
23.6	Identifying Class Operations	839
23.7	Indicating Collaboration Among Objects	845
23.8	Wrap-Up	852

24	ATM Case Study Part 2: Implementing an Object-Oriented Design	856
24.1	Introduction	857
24.2	Starting to Program the Classes of the ATM System	857
24.3	Incorporating Inheritance and Polymorphism into the ATM System	862
24.4	ATM Case Study Implementation	868
24.4.1	Class ATM	869
24.4.2	Class Screen	874
24.4.3	Class Keypad	875
24.4.4	Class CashDispenser	876
24.4.5	Class DepositSlot	877
24.4.6	Class Account	878
24.4.7	Class BankDatabase	880
24.4.8	Class Transaction	883
24.4.9	Class BalanceInquiry	884
24.4.10	Class Withdrawal	885
24.4.11	Class Deposit	889
24.4.12	Class ATMCaseStudy	892
24.5	Wrap-Up	893
A	Operator Precedence Chart	895
B	ASCII Character Set	897
C	Keywords and Reserved Words	898
D	Primitive Types	899
E	Using the Debugger	900
E.1	Introduction	901
E.2	Breakpoints and the run, stop, cont and print Commands	901
E.3	The print and set Commands	905

E.4	Controlling Execution Using the <code>step</code> , <code>step up</code> and <code>next</code> Commands	907
E.5	The <code>watch</code> Command	909
E.6	The <code>clear</code> Command	912
E.7	Wrap-Up	914
F	Using the Java API Documentation	915
F.1	Introduction	915
F.2	Navigating the Java API	916
G	Creating Documentation with <code>javadoc</code>	924
G.1	Introduction	924
G.2	Documentation Comments	924
G.3	Documenting Java Source Code	925
G.4	<code>javadoc</code>	932
G.5	Files Produced by <code>javadoc</code>	933
H	Unicode®	937
H.1	Introduction	937
H.2	Unicode Transformation Formats	938
H.3	Characters and Glyphs	939
H.4	Advantages/Disadvantages of Unicode	940
H.5	Using Unicode	940
H.6	Character Ranges	942
I	Formatted Output	944
I.1	Introduction	945
I.2	Streams	945
I.3	Formatting Output with <code>printf</code>	945
I.4	Printing Integers	946
I.5	Printing Floating-Point Numbers	947
I.6	Printing Strings and Characters	949
I.7	Printing Dates and Times	950
I.8	Other Conversion Characters	952
I.9	Printing with Field Widths and Precisions	954
I.10	Using Flags in the <code>printf</code> Format String	956
I.11	Printing with Argument Indices	960
I.12	Printing Literals and Escape Sequences	960
I.13	Formatting Output with Class <code>Formatter</code>	961
I.14	Wrap-Up	962

J	Number Systems	963
J.1	Introduction	964
J.2	Abbreviating Binary Numbers as Octal and Hexadecimal Numbers	967
J.3	Converting Octal and Hexadecimal Numbers to Binary Numbers	968
J.4	Converting from Binary, Octal or Hexadecimal to Decimal	968
J.5	Converting from Decimal to Binary, Octal or Hexadecimal	969
J.6	Negative Binary Numbers: Two's Complement Notation	971
K	Bit Manipulation	973
K.1	Introduction	973
K.2	Bit Manipulation and the Bitwise Operators	973
K.3	BitSet Class	983
L	Labeled break and continue Statements	987
L.1	Introduction	987
L.2	Labeled break Statement	987
L.3	Labeled continue Statement	988
M	UML 2: Additional Diagram Types	990
M.1	Introduction	990
M.2	Additional Diagram Types	990
N	Design Patterns	992
N.1	Introduction	992
N.2	Creational, Structural and Behavioral Design Patterns	993
	N.2.1 Creational Design Patterns	994
	N.2.2 Structural Design Patterns	996
	N.2.3 Behavioral Design Patterns	997
	N.2.4 Conclusion	998
N.3	Design Patterns in Packages <code>java.awt</code> and <code>javax.swing</code>	998
	N.3.1 Creational Design Patterns	999
	N.3.2 Structural Design Patterns	999
	N.3.3 Behavioral Design Patterns	1001
	N.3.4 Conclusion	1005
N.4	Concurrency Design Patterns	1005
N.5	Design Patterns Used in Packages <code>java.io</code> and <code>java.net</code>	1006
	N.5.1 Creational Design Patterns	1006
	N.5.2 Structural Design Patterns	1006
	N.5.3 Architectural Patterns	1008
	N.5.4 Conclusion	1010
N.6	Design Patterns Used in Package <code>java.util</code>	1010

N.6.1	Creational Design Patterns	1010
N.6.2	Behavioral Design Patterns	1010
N.7	Wrap-Up	1011

Index **1013**

This page intentionally left blank

Foreword

I've been enamored with Java even prior to its 1.0 release in 1995, and have subsequently been a Java developer, author, speaker, teacher and Oracle Java Technology Ambassador. In this journey, it has been my privilege to call Paul Deitel a colleague, and to often leverage and recommend his Java books. In their many editions, these books have proven to be great texts for college and professional courses that I and others have developed to teach the Java programming language.

One of the qualities that makes *Java SE 8 for Programmers, 3/e*, a great resource is its thorough and insightful coverage of Java concepts. Another useful quality is its treatment of concepts and practices essential to effective software development.

I'd like to point out some of the features of this new edition about which I'm most excited:

- An ambitious new chapter on Java lambda expressions and streams. This chapter starts out with a primer on functional programming, and introduces Java lambda expressions and how to use streams to perform functional programming tasks on collections.
- Although concurrency has been addressed since the first edition of the book, it is increasingly important because of multi-core architectures. There are timing examples—using the new Date/Time API classes introduced in Java SE 8—in the concurrency chapter that show the performance improvements with multi-core over single-core.
- JavaFX is Java's GUI/graphics/multimedia technology moving forward, so it is nice to see JavaFX introduced in the Deitel live-code pedagogic style.

Please join me in congratulating Paul and Harvey Deitel on their latest edition of a wonderful resource for software developers!

James L. Weaver
Java Technology Ambassador
Oracle Corporation

This page intentionally left blank

Preface

Welcome to Java and *Java SE 8 for Programmers, Third Edition!* This book presents leading-edge computing technologies for software developers.

We focus on software engineering best practices. At the heart of the book is the Deitel signature “live-code approach”—rather than using code snippets, we present concepts in the context of complete working programs that run on recent versions of Windows®, Linux® and OS X®. Each complete code example is accompanied by live sample executions. All the source code is available at

<http://www.deitel.com/books/javafp3/>

Keeping in Touch with the Authors

As you read the book, if you have questions, send an e-mail to us at

deitel@deitel.com

and we'll respond promptly. For updates on this book, visit

<http://www.deitel.com/books/jfp3>

subscribe to the *Deitel® Buzz Online* newsletter at

<http://www.deitel.com/newsletter/subscribe.html>

and join the Deitel social networking communities on

- Facebook® (<http://www.deitel.com/deitelfan>)
- Twitter® (@deitel)
- Google+™ (<http://google.com/+DeitelFan>)
- YouTube® (<http://youtube.com/DeitelTV>)
- LinkedIn® (<http://linkedin.com/company/deitel-&-associates>)

Modular Organization

Java SE 8 for Programmers, 3/e, is appropriate for programmers with a background in high-level language programming. It features a modular organization:

Introduction

- Chapter 1, Introduction to Java and Test-Driving a Java Application
- Chapter 2, Introduction to Java Applications; Input/Output and Operators
- Chapter 3, Introduction to Classes, Objects, Methods and Strings

Additional Programming Fundamentals

- Chapter 4, Control Statements: Part 1; Assignment, `++` and `--` Operators
- Chapter 5, Control Statements: Part 2; Logical Operators
- Chapter 6, Methods: A Deeper Look
- Chapter 7, Arrays and ArrayLists
- Chapter 14, Strings, Characters and Regular Expressions
- Chapter 15, Files, Streams and Object Serialization

Object-Oriented Programming

- Chapter 8, Classes and Objects: A Deeper Look
- Chapter 9, Object-Oriented Programming: Inheritance
- Chapter 10, Object-Oriented Programming: Polymorphism and Interfaces
- Chapter 11, Exception Handling: A Deeper Look

Swing and JavaFX Graphical User Interfaces; Java 2D Graphics

- Chapter 12, Swing GUI Components: Part 1
- Chapter 13, Graphics and Java 2D
- Chapter 19, Swing GUI Components: Part 2
- Chapter 22, JavaFX GUI

Generic Collections, Lambdas and Streams

- Chapter 16, Generic Collections
- Chapter 17, Java SE 8 Lambdas and Streams
- Chapter 18, Generic Classes and Methods

Concurrency/Database

- Chapter 20, Concurrency
- Chapter 21, Accessing Databases with JDBC

Object-Oriented Design

- Chapter 23, ATM Case Study, Part 1: Object-Oriented Design with the UML
- Chapter 24, ATM Case Study Part 2: Implementing an Object-Oriented Design

New and Updated Features

Here are the updates we've made for *Java SE 8 for Programmers, 3/e*:

- *Easy to use with Java SE 7 or Java SE 8.* This book was published coincident with the release of Java SE 8. To meet the needs of our diverse audiences, we designed the book for professionals interested in Java SE 7, Java SE 8 or a mixture

of both. The Java SE 8 features (Fig. 4.1) are covered in Chapter 17 and in easy-to-include-or-omit sections book wide.

Java SE 8 features
Lambda expressions
Type-inference improvements
@FunctionalInterface annotation
Parallel array sorting
Bulk data operations for Java Collections— <code>filter</code> , <code>map</code> and <code>reduce</code>
Library enhancements to support lambdas (e.g., <code>java.util.stream</code> , <code>java.util.function</code>)
Date & Time API (<code>java.time</code>)
Java concurrency API improvements
<code>static</code> and <code>default</code> methods in interfaces
Functional interfaces—interfaces that define only one <code>abstract</code> method and can include <code>static</code> and <code>default</code> methods
JavaFX enhancements

Fig. 4.1 | Java SE 8 features we discuss.

- *Java SE 8 lambdas, streams, and interfaces with default and static methods.* The most significant new features in Java SE 8 are lambdas and complementary technologies. In Chapter 17, you’ll see that functional programming with lambdas and streams can help you write programs faster, more concisely, more simply, with fewer bugs and that are easier to parallelize (to get performance improvements on multi-core systems) than programs written with previous techniques (Fig. 4.2). You’ll see that functional programming complements object-oriented programming.

Pre-Java-SE-8 topics	Corresponding Java SE 8 discussions and examples
Chapter 7, Arrays and ArrayLists	Sections 17.3–17.4 introduce basic lambda and streams capabilities that process one-dimensional arrays.
Chapter 10, Object-Oriented Programming: Polymorphism and Interfaces	Section 10.10 introduces the new Java SE 8 interface features (<code>default</code> methods, <code>static</code> methods and the concept of functional interfaces) that support functional programming with lambdas and streams.
Chapters 12 and 19, Swing GUI Components: Parts 1 and 2	Section 17.9 shows how to use a lambda to implement a Swing event-listener functional interface.
Chapter 14, Strings, Characters and Regular Expressions	Section 17.5 shows how to use lambdas and streams to process collections of <code>String</code> objects.

Fig. 4.2 | Java SE 8 lambdas and streams discussions and examples. (Part I of 2.)

Pre-Java-SE-8 topics	Corresponding Java SE 8 discussions and examples
Chapter 15, Files, Streams and Object Serialization	Section 17.7 shows how to use lambdas and streams to process lines of text from a file.
Chapter 20, Concurrency	Shows that functional programs are easier to parallelize so that they can take advantage of multi-core architectures to enhance performance. Demonstrates parallel stream processing. Shows that <code>Arrays</code> method <code>parallelSort</code> improves performance on multi-core architectures when sorting large arrays.
Chapter 22, JavaFX GUI	Section 22.5.5 shows how to use a lambda to implement a JavaFX event-listener functional interface.

Fig. 4.2 | Java SE 8 lambdas and streams discussions and examples. (Part 2 of 2.)

- *Java SE 7's try-with-resources statement and the AutoClosable interface.* AutoClosable objects reduce the likelihood of resource leaks when you use them with the try-with-resources statement, which automatically closes the AutoClosable objects. In this edition, we use try-with-resources and AutoClosable objects as appropriate starting in Chapter 15, Files, Streams and Object Serialization.
- *Java security.* We audited our book against the CERT Oracle Secure Coding Standard for Java:

<http://bit.ly/CERT0oracleSecureJava>

See this Preface's Secure Java Programming section for more about CERT.

- *Java NIO API.* We updated the file-processing examples in Chapter 15 to use features from the Java NIO (new IO) API.
- *Java Documentation.* Throughout the book, we provide links to Java documentation where you can learn more about various topics that we present. For Java SE 7 documentation, the links begin with

<http://docs.oracle.com/javase/7/>

and for Java SE 8 documentation, the links begin with

<http://download.java.net/jdk8/>

These links could change when Oracle releases Java SE 8—possibly to links beginning with

<http://docs.oracle.com/javase/8/>

For any links that change after publication, we'll post updates at

<http://www.deitel.com/books/jfp3>

Swing and JavaFX GUI; Java 2D Graphics

- *Swing GUI and Java 2D graphics.* Java's Swing GUI is discussed in Chapters 12 and 19. Swing is now in maintenance mode—Oracle has stopped development

and will provide only bug fixes going forward, however it will remain part of Java and is still widely used. Most of GUI-based legacy code in industry uses Swing GUI. Chapter 13 discusses Java 2D graphics.

- **JavaFX GUI.** Java's GUI, graphics and multimedia technology going forward is JavaFX. In Chapter 22, we use JavaFX 2.2 with Java SE 7. We use Scene Builder—a drag-and-drop tool for creating JavaFX GUIs quickly and conveniently. It's a standalone tool that you can use separately or with Java IDEs.

Concurrency

- **Concurrency for optimal multi-core performance.** In this edition, we were privileged to have as a reviewer Brian Goetz, co-author of *Java Concurrency in Practice* (Addison-Wesley). We updated Chapter 20, Concurrency, with Java SE 8 technology and idiom. We added a `parallelSort` vs. `sort` example that uses the Java SE 8 Date/Time API to time each operation and demonstrate `parallelSort`'s better performance on a multi-core system. We include a Java SE 8 parallel vs. sequential stream processing example, again using the Date/Time API to show performance improvements. Finally, we added a Java SE 8 `CompletableFuture` example that compares the relative performance of sequential and parallel execution of long-running calculations.
- **SwingWorker class.** We use class `SwingWorker` to create multithreaded user interfaces.
- **Concurrency is challenging.** There's a great variety of concurrency features. We point out the ones that most developers should use and mention those that should be left to the experts.

Getting Monetary Amounts Right

- **Monetary amounts.** In the early chapters, for convenience, we use type `double` to represent monetary amounts. Due to the potential for incorrect monetary calculations with type `double`, class `BigDecimal` (which is a bit more complex) should be used to represent monetary amounts. We demonstrate `BigDecimal` in Chapters 8 and 22.

Object Technology

- **Object-oriented programming.** We use an *early objects* approach, reviewing the basic concepts and terminology of object technology in Chapter 1. Readers develop their first customized classes and objects in Chapter 3.
- **Early objects real-world case studies.** The early classes and objects presentation features `Account`, `Student`, `AutoPolicy`, `Time`, `Employee`, `GradeBook` and `Card` shuffling-and-dealing case studies, gradually introducing deeper OO concepts.
- **Inheritance, Interfaces, Polymorphism and Composition.** We use a series of real-world case studies to illustrate each of these OO concepts and explain situations in which each is preferred in building industrial-strength applications. We discuss Java SE 8's improvements to the interface concept.

- **Exception handling.** We integrate basic exception handling early in the book then present a deeper treatment in Chapter 11. Exception handling is important for building “mission-critical” and “business-critical” applications. Programmers need to be concerned with, “What happens when the component I call on to do a job experiences difficulty? How will that component signal that it had a problem?” To use a Java component, you need to know not only how that component behaves when “things go well,” but also what exceptions that component “throws” when “things go poorly.”
- **Class Arrays and ArrayList.** Chapter 7 covers class `Arrays`—which contains methods for performing common array manipulations—and class `ArrayList`—which implements a dynamically resizable array-like data structure. This follows our philosophy of getting lots of practice using existing classes while learning how to define your own classes.
- **Case Study: Developing an Object-Oriented Design and Java Implementation of an ATM.** Chapters 23–24 include a case study on object-oriented design with the UML (Unified Modeling Language™)—the industry-standard graphical language for modeling object-oriented systems. We design and implement the software for a simple automated teller machine (ATM). We analyze a typical requirements document that specifies the system to be built. We determine the classes needed to implement that system, the attributes the classes need to have, the behaviors the classes need to exhibit and specify how the classes must interact with one another to meet the system requirements. From the design we produce a completely coded Java implementation. Participants in our professional Java courses often report having a “light-bulb moment”—the case study helps them “tie it all together” and really understand Java-based object-oriented programming.

Generic Collections

- **Generic collections presentation.** We begin with generic class `ArrayList` in Chapter 7. Chapters 16–18 provide a deeper treatment of generic collections—showing how to use the built-in collections of the Java API. We show how to implement generic methods and classes. Lambdas and streams (introduced in Chapter 17) are especially useful for working with generic collections.

Database

- **JDBC.** Chapter 21 covers JDBC and uses the Java DB database management system. The chapter introduces Structured Query Language (SQL) and features an OO case study on developing a database-driven address book that demonstrates prepared statements.

Secure Java Programming

It’s difficult to build industrial-strength systems that stand up to attacks from viruses, worms, and other forms of “malware.” Today, via the Internet, such attacks can be instantaneous and global in scope. Building security into software from the beginning of the development cycle can greatly reduce vulnerabilities. We incorporate various secure Java coding practices into our discussions and code examples.

The CERT® Coordination Center (www.cert.org) was created to analyze and respond promptly to attacks. CERT—the Computer Emergency Response Team—is a government-funded organization within the Carnegie Mellon University Software Engineering Institute™. CERT publishes and promotes secure coding standards for various popular programming languages to help software developers implement industrial-strength systems that avoid the programming practices which leave systems open to attack.

We'd like to thank Robert C. Seacord, Secure Coding Manager at CERT and an adjunct professor in the Carnegie Mellon University School of Computer Science. Mr. Seacord was a technical reviewer for our book, *C11 for Programmers*, where he scrutinized our C programs from a security standpoint, recommending that we adhere to the *CERT C Secure Coding Standard*. This experience influenced our coding practices in *C++11 for Programmers* and *Java SE 8 for Programmers, 3/e* as well.

Teaching Approach

Java SE 8 for Programmers, 3/e, contains hundreds of complete working examples. We stress program clarity and concentrate on building well-engineered software.

Syntax Coloring. For readability, we syntax color the code, similar to the way most integrated-development environments and code editors syntax color the code. Our syntax-coloring conventions are:

```
comments appear like this  
keywords appear like this  
constants and literal values appear like this  
errors appear like this  
all other code appears in black
```

Code Highlighting. We place yellow rectangles around each program's key code.

Using Fonts for Emphasis. We place the key terms and the index's page reference for each defining occurrence in **bold** text for easier reference. On-screen components are emphasized in the **bold Helvetica** font (e.g., the **File** menu) and Java program text in the Lucida font (e.g., `int x = 5;`).

Web Access. All of the source-code examples can be downloaded from:

```
www.deitel.com/books/javafp3  
www.pearsonhighered.com/deitel
```

Objectives. The opening quotations are followed by a list of chapter objectives.

Illustrations/Figures. Abundant tables, line drawings, UML diagrams, programs and program outputs are included.

Programming Tips. We include programming tips to help you focus on important aspects of program development. These tips and practices represent the best we've gleaned from a combined seven decades of programming and teaching experience.



Good Programming Practice

The Good Programming Practices call attention to techniques that will help you produce programs that are clearer, more understandable and more maintainable.



Common Programming Error

Pointing out these Common Programming Errors reduces the likelihood that you'll make them.



Error-Prevention Tip

These tips contain suggestions for exposing and removing bugs from your programs; many of the tips describe aspects of Java that prevent bugs from getting into programs.



Performance Tip 4.1

These tips highlight opportunities for making your programs run faster or minimizing the amount of memory that they occupy.



Portability Tip

The Portability Tips help you write code that will run on a variety of platforms.



Software Engineering Observation

The Software Engineering Observations highlight architectural and design issues that affect the construction of software systems, especially large-scale systems.



Look-and-Feel Observation

The Look-and-Feel Observations highlight graphical-user-interface conventions. These observations help you design attractive, user-friendly graphical user interfaces that conform to industry norms.

Index. We've included an extensive index. Defining occurrences of key terms are highlighted with a bold page number.

Software Used in *Java SE 8 for Programmers, 3/e*

All the software you'll need for this book is available free for download from the Internet. See the Before You Begin section that follows this Preface for links to each download.

We wrote most of the examples in *Java SE 8 for Programmers, 3/e*, using the free Java Standard Edition Development Kit (JDK) 7. For the Java SE 8 modules, we used the OpenJDK's early access version of JDK 8. In Chapter 22, we also used the Netbeans IDE. See the Before You Begin section that follows this Preface for more information.

Java Fundamentals: Parts I, II, III and IV LiveLessons, Second Edition, Video Training Product

Our *Java Fundamentals: Parts I, II, III and IV LiveLessons, 2/e* (summer 2014), video training product shows you what you need to know to start building robust, powerful software with Java. It includes 30+ hours of expert training synchronized with *Java SE 8 for Programmers, Third Edition*. Visit

<http://www.deitel.com/livelessons>

for information on purchasing Deitel LiveLessons video products online from Informit and Udemy. You may also access our LiveLessons videos if you have a subscription to Safari Books Online (<http://www.safaribooksonline.com>).

Acknowledgments

We'd like to thank Abbey Deitel and Barbara Deitel of Deitel & Associates, Inc. for long hours devoted to this project. Abbey co-authored Chapter 1 and this Preface, and she and Barbara painstakingly researched the new capabilities of Java SE 8.

We're fortunate to have worked on this project with the dedicated publishing professionals at Prentice Hall/Pearson. We appreciate the extraordinary efforts and 19-year mentorship of our friend and professional colleague Mark L. Taub, Editor-in-Chief of Pearson Technology Group. Carole Snyder recruited distinguished members of the Java community to review the manuscript and managed the review process. Chuti Prasertsith designed the cover. John Fuller managed the book's publication.

Reviewers

We wish to acknowledge the efforts of our recent editions reviewers—a distinguished group of Oracle Java team members, Oracle Java Champions, other industry professionals and academics. They scrutinized the text and the programs and provided countless suggestions for improving the presentation.

Third Edition reviewers: Lance Andersen (Oracle Corporation), Dr. Danny Coward (Oracle Corporation), Brian Goetz (Oracle Corporation), Evan Golub (University of Maryland), Dr. Huiwei Guan (Professor, Department of Computer & Information Science, North Shore Community College), Manfred Riem (Java Champion), Simon Ritter (Oracle Corporation), Robert C. Seacord (CERT, Software Engineering Institute, Carnegie Mellon University), Khallai Taylor (Assistant Professor, Triton College and Adjunct Professor, Lonestar College—Kingwood), Jorge Vargas (Yumbling and a Java Champion), Johan Vos (LodgON and Oracle Java Champion) and James L. Weaver (Oracle Corporation and author of *Pro JavaFX 2*).

Other recent editions reviewers: Soundararajan Angusamy (Sun Microsystems), Joseph Bowbeer (Consultant), William E. Duncan (Louisiana State University), Diana Franklin (University of California, Santa Barbara), Edward F. Gehringer (North Carolina State University), Ric Heishman (George Mason University), Dr. Heinz Kabutz (JavaSpecialists.eu), Patty Kraft (San Diego State University), Lawrence Premkumar (Sun Microsystems), Tim Margush (University of Akron), Sue McFarland Metzger (Villanova University), Shyamal Mitra (The University of Texas at Austin), Peter Pilgrim (Consultant), Manjeet Rege, Ph.D. (Rochester Institute of Technology), Susan Rodger (Duke University), Amr Sabry (Indiana University), José Antonio González Seco (Parliament of Andalusia), Sang Shin (Sun Microsystems), S. Sivakumar (Astra Infotech Private Limited), Raghavan “Rags” Srinivas (Intuit), Monica Sweat (Georgia Tech), Vinod Varma (Astra Infotech Private Limited) and Alexander Zuev (Sun Microsystems).

A Special Thank You to Brian Goetz

We were privileged to have Brian Goetz, Oracle's Java Language Architect and Specification Lead for Java SE 8's Project Lambda, and co-author of *Java Concurrency in Practice*, do a detailed full-book review. He thoroughly scrutinized every chapter, providing extremely helpful insights and constructive comments. Any remaining faults in the book are our own.

Well, there you have it! As you read the book, we'd appreciate your comments, criticisms, corrections and suggestions for improvement. Please address all correspondence to:

deitel@deitel.com

We'll respond promptly. We hope you enjoy working with *Java SE 8 for Programmers, 3/e*, as much as we enjoyed writing it!

Paul and Harvey Deitel

About the Authors



Paul Deitel, CEO and Chief Technical Officer of Deitel & Associates, Inc., is a graduate of MIT, where he studied Information Technology. He holds the Java Certified Programmer and Java Certified Developer designations, and is an Oracle Java Champion. Through Deitel & Associates, Inc., he has delivered hundreds of programming courses worldwide to clients, including Cisco, IBM, Siemens, Sun Microsystems, Dell, Fidelity, NASA at the Kennedy Space Center, the National Severe Storm Laboratory, White Sands Missile Range, Rogue Wave Software, Boeing, SunGard Higher Education, Nortel Networks, Puma, iRobot, Invensys and many more. He and his co-author, Dr. Harvey M. Deitel, are the world's best-selling programming-language textbook/professional book/video authors.

Dr. Harvey Deitel, Chairman and Chief Strategy Officer of Deitel & Associates, Inc., has over 50 years of experience in the computer field. Dr. Deitel earned B.S. and M.S. degrees in Electrical Engineering from MIT and a Ph.D. in Mathematics from Boston University. He has extensive college teaching experience, including earning tenure and serving as the Chairman of the Computer Science Department at Boston College before founding Deitel & Associates, Inc., in 1991 with his son, Paul. The Deitels' publications have earned international recognition, with translations published in Japanese, German, Russian, Spanish, French, Polish, Italian, Simplified Chinese, Traditional Chinese, Korean, Portuguese, Greek, Urdu and Turkish. Dr. Deitel has delivered hundreds of programming courses to corporate, academic, government and military clients.

About Deitel® & Associates, Inc.

Deitel & Associates, Inc., founded by Paul Deitel and Harvey Deitel, is an internationally recognized authoring and corporate training organization, specializing in computer programming languages, object technology, mobile app development and Internet and web software technology. The company's training clients include many of the world's largest companies, government agencies, branches of the military, and academic institutions. The company offers instructor-led training courses delivered at client sites worldwide on major programming languages and platforms, including Java™, Android app development, Objective-C and iOS app development, C++, C, Visual C#®, Visual Basic®, Visual C++®, Python®, object technology, Internet and web programming and a growing list of additional programming and software development courses.

Through its 39-year publishing partnership with Pearson/Prentice Hall, Deitel & Associates, Inc., publishes leading-edge programming textbooks and professional books in print and a wide range of e-book formats, and *LiveLessons* video courses. Deitel & Associates, Inc. and the authors can be reached at:

deitel@deitel.com

To learn more about Deitel's *Dive-Into® Series* Corporate Training curriculum, visit:

<http://www.deitel.com/training>

To request a proposal for worldwide on-site, instructor-led training at your organization, e-mail deitel@deitel.com.

Individuals wishing to purchase Deitel books and *LiveLessons* video training can do so through www.deitel.com. Bulk orders by corporations, the government, the military and academic institutions should be placed directly with Pearson. For more information, visit

<http://www.informit.com/store/sales.aspx>

This page intentionally left blank

Before You Begin

This section contains information you should review before using this book. Any updates to the information presented here will be posted at:

<http://www.deitel.com/books/javafp3>

In addition, we provide Dive-Into® videos (which will be available in time for Fall 2014 classes) that demonstrate the instructions in this Before You Begin section.

Font and Naming Conventions

We use fonts to distinguish between on-screen components (such as menu names and menu items) and Java code or commands. Our convention is to emphasize on-screen components in a sans-serif bold **Helvetica** font (for example, **File** menu) and to emphasize Java code and commands in a sans-serif **Lucida** font (for example, **System.out.println()**).

Software Used in the Book

All the software you'll need for this book is available free for download from the web. With the exception of the examples that are specific to Java SE 8, all of the examples were tested with the Java SE 7 and Java SE 8 Java Standard Edition Development Kits (JDKs).

Java Standard Edition Development Kit 7 (JDK 7)

JDK 7 for Windows, OS X and Linux platforms is available from:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Java Standard Edition Development Kit (JDK) 8

At the time of this publication, the near-final version of JDK 8 for Windows, OS X and Linux platforms was available from:

<https://jdk8.java.net/download.html>

Once JDK 8 is released as final, it will be available from:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

JDK Installation Instructions

After downloading the JDK installer, be sure to carefully follow the JDK installation instructions for your platform at:

<http://docs.oracle.com/javase/7/docs/webnotes/install/index.html>

Though these instructions are for JDK 7, they also apply to JDK 8—you'll need to update the JDK version number in any version-specific instructions.

Setting the PATH Environment Variable

The PATH environment variable on your computer designates which directories the computer searches when looking for applications, such as the applications that enable you to compile and run your Java applications (called javac and java, respectively). *Carefully follow the installation instructions for Java on your platform to ensure that you set the PATH environment variable correctly.* The steps for setting environment variables differ by operating system and sometimes by operating system version (e.g., Windows 7 vs. Windows 8). Instructions for various platforms are listed at:

```
http://www.java.com/en/download/help/path.xml
```

If you do not set the PATH variable correctly on Windows and some Linux installations, when you use the JDK's tools, you'll receive a message like:

```
'java' is not recognized as an internal or external command,  
operable program or batch file.
```

In this case, go back to the installation instructions for setting the PATH and recheck your steps. If you've downloaded a newer version of the JDK, you may need to change the name of the JDK's installation directory in the PATH variable.

JDK Installation Directory and the bin Subdirectory

The JDK's installation directory varies by platform. The directories listed below are for Oracle's JDK 7 update 51:

- 32-bit JDK on Windows:
`C:\Program Files (x86)\Java\jdk1.7.0_51`
- 64-bit JDK on Windows:
`C:\Program Files\Java\jdk1.7.0_51`
- Mac OS X:
`/Library/Java/JavaVirtualMachines/jdk1.7.0_51.jdk/Contents/Home`
- Ubuntu Linux:
`/usr/lib/jvm/java-7-oracle`

Depending on your platform, the JDK installation folder's name might differ if you're using a different update of JDK 7 or using JDK 8. For Linux, the install location depends on the installer you use and possibly the version of Linux that you use. We used Ubuntu Linux. The PATH environment variable must point to the JDK installation directory's bin subdirectory.

When setting the PATH, be sure to use the proper JDK-installation-directory name for the specific version of the JDK you installed—as newer JDK releases become available, the JDK-installation-directory name changes to include an *update version number*. For example, at the time of this writing, the most recent JDK 7 release was update 51. For this version, the JDK-installation-directory name ends with `_51`.

Setting the CLASSPATH Environment Variable

If you attempt to run a Java program and receive a message like

```
Exception in thread "main" java.lang.NoClassDefFoundError: YourClass
```

then your system has a CLASSPATH environment variable that must be modified. To fix the preceding error, follow the steps in setting the PATH environment variable to locate the CLASSPATH variable, then edit the variable's value to include the local directory—typically represented as a dot (.). On Windows add

```
. ;
```

at the beginning of the CLASSPATH's value (with no spaces before or after these characters). On other platforms, replace the semicolon with the appropriate path separator characters—typically a colon (:).

Setting the JAVA_HOME Environment Variable

The Java DB database software that you'll use in Chapter 21 requires you to set the JAVA_HOME environment variable to your JDK's installation directory. The same steps you used to set the PATH may also be used to set other environment variables, such as JAVA_HOME.

Java Integrated Development Environments (IDEs)

There are many Java integrated development environments that you can use for Java programming. For this reason, we used only the JDK command-line tools for most of the book's examples. We provide Dive-Into® videos (which will be available in time for Fall 2014 classes) that show how to download, install and use three popular IDEs—NetBeans, Eclipse and IntelliJ IDEA. We use NetBeans in Chapter 22.

NetBeans Downloads

You can download the JDK/NetBeans bundle from:

```
http://www.oracle.com/technetwork/java/javase/downloads/index.html
```

The NetBeans version that's bundled with the JDK is for Java SE development. The online JavaServer Faces (JSF) chapters and web services chapter use the Java Enterprise Edition (Java EE) version of NetBeans, which you can download from:

```
https://netbeans.org/downloads/
```

This version supports both Java SE and Java EE development.

Eclipse Downloads

You can download the Eclipse IDE from:

```
https://www.eclipse.org/downloads/
```

For Java SE development choose the Eclipse IDE for Java Developers. For Java Enterprise Edition (Java EE) development (such as JSF and web services), choose the Eclipse IDE for Java EE Developers—this version supports both Java SE and Java EE development.

IntelliJ IDEA Community Edition Downloads

You can download the free IntelliJ IDEA Community Edition from:

```
http://www.jetbrains.com/idea/download/index.html
```

The free version supports only Java SE development.

Obtaining the Code Examples

The examples for *Java SE 8 for Programmers, 3/e* are available for download at

<http://www.deitel.com/books/javafp3>

under the heading **Download Code Examples and Other Premium Content**. The examples are also available from

<http://www.pearsonhighered.com/deitel>

When you download the ZIP archive file, write down the location where you choose to save it on your computer.

Extract the contents of `examples.zip` using a ZIP extraction tool such as 7-Zip (www.7-zip.org), WinZip (www.winzip.com) or the built-in capabilities of your operating system. Instructions throughout the book assume that the examples are located at:

- C:\examples on Windows
- your user account home folder's examples subfolder on Linux
- your Documents folders examples subfolder on Mac OS X

Java's Nimbus Look-and-Feel

Java comes bundled with a cross-platform look-and-feel known as Nimbus. For programs with Swing graphical user interfaces (e.g., Chapters 12 and 19), we configured our test computers to use Nimbus as the default look-and-feel.

To set Nimbus as the default for all Java applications, you must create a text file named `swing.properties` in the `lib` folder of both your JDK installation folder and your JRE installation folder. Place the following line of code in the file:

`swing.defaultlaf=com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel`

For more information on locating these folders visit <http://docs.oracle.com/javase/7/docs/webnotes/install/index.html>. [Note: In addition to the standalone JRE, there's a JRE nested in your JDK's installation folder. If you're using an IDE that depends on the JDK (e.g., NetBeans), you may also need to place the `swing.properties` file in the nested `jre` folder's `lib` folder.]

You're now ready to begin your Java studies with *Java SE 8 for Programmers, 3/e*. We hope you enjoy the book!

3

Introduction to Classes, Objects, Methods and Strings

Objectives

In this chapter you'll:

- Declare a class and use it to create an object.
- Implement a class's behaviors as methods.
- Implement a class's attributes as instance variables.
- Call an object's methods to make them perform their tasks.
- Understand how local variables of a method differ from instance variables.
- Understand what primitive types and reference types are.
- Use a constructor to initialize an object's data.

Outline

3.1 Introduction	3.3 Primitive Types vs. Reference Types
3.2 Instance Variables, <i>set</i> Methods and <i>get</i> Methods	3.4 Account Class: Initializing Objects with Constructors
3.2.1 Account Class with an Instance Variable, a <i>set</i> Method and a <i>get</i> Method	3.4.1 Declaring an Account Constructor for Custom Object Initialization
3.2.2 AccountTest Class That Creates and Uses an Object of Class Account	3.4.2 Class AccountTest: Initializing Account Objects When They're Created
3.2.3 Compiling and Executing an App with Multiple Classes	3.5 Account Class with a Balance; Floating-Point Numbers
3.2.4 Account UML Class Diagram with an Instance Variable and <i>set</i> and <i>get</i> Methods	3.5.1 Account Class with a balance Instance Variable of Type double
3.2.5 Additional Notes on Class AccountTest	3.5.2 AccountTest Class to Use Class Account
3.2.6 Software Engineering with private Instance Variables and public <i>set</i> and <i>get</i> Methods	3.6 Wrap-Up

3.1 Introduction

[*Note:* This chapter depends on the terminology and concepts discussed in Section 1.2, Object Technology Concepts.]

In Chapter 2, you worked with existing classes, objects and methods. You used the pre-defined standard output object `System.out`, invoking its methods `print`, `println` and `printf` to display information on the screen. You used the existing `Scanner` class to create an object that reads into memory integer data typed by the user at the keyboard. Throughout the book, you'll use many more preexisting classes and objects.

In this chapter, you'll create your own classes and methods. Each new class you create becomes a new type that can be used to declare variables and create objects. You can declare new classes as needed; this is one reason why Java is known as an *extensible* language.

We present a case study on creating and using a simple, real-world bank account class—`Account`. Such a class should maintain as *instance variables* attributes such as its `name` and `balance`, and provide *methods* for tasks such as querying the balance (`getBalance`), making deposits that increase the balance (`deposit`) and making withdrawals that decrease the balance (`withdraw`). We'll build the `getBalance` and `deposit` methods into the class in the chapter's examples.

In Chapter 2 we used the data type `int` to represent integers. In this chapter, we introduce data type `double` to represent an account balance as a number that can contain a decimal *point*—such numbers are called floating-point numbers. [In Chapter 8, when we get a bit deeper into object technology, we'll begin representing monetary amounts precisely with class `BigDecimal` (package `java.math`) as you should do when writing industrial-strength monetary applications.]

3.2 Instance Variables, *set* Methods and *get* Methods

In this section, you'll create two classes—`Account` (Fig. 3.1) and `AccountTest` (Fig. 3.2). Class `AccountTest` is an *application class* in which the `main` method will create and use an `Account` object to demonstrate class `Account`'s capabilities.

3.2.1 Account Class with an Instance Variable, a set Method and a get Method

Different accounts typically have different names. For this reason, class `Account` (Fig. 3.1) contains a `name` *instance variable*. A class's instance variables maintain data for each object (that is, each instance) of the class. Later in the chapter we'll add an instance variable named `balance` so we can keep track of how much money is in the account. Class `Account` contains two methods—method `setName` stores a name in an `Account` object and method `getName` obtains a name from an `Account` object.

```

1 // Fig. 3.1: Account.java
2 // Account class that contains a name instance variable
3 // and methods to set and get its value.
4
5 public class Account
6 {
7     private String name; // instance variable
8
9     // method to set the name in the object
10    public void setName(String name)
11    {
12        this.name = name; // store the name
13    }
14
15    // method to retrieve the name from the object
16    public String getName()
17    {
18        return name; // return value of name to caller
19    }
20 } // end class Account

```

Fig. 3.1 | `Account` class that contains a `name` instance variable and methods to *set* and *get* its value.

Class Declaration

The *class declaration* begins in line 5. The keyword `public` (which Chapter 8 explains in detail) is an *access modifier*. For now, we'll simply declare every class `public`. Each `public` class declaration must be stored in a file having the *same* name as the class and ending with the `.java` filename extension; otherwise, a compilation error will occur. Thus, `public` classes `Account` and `AccountTest` (Fig. 3.2) *must* be declared in the *separate* files `Account.java` and `AccountTest.java`, respectively.

Every class declaration contains the keyword `class` followed immediately by the class's name—in this case, `Account`. Every class's body is enclosed in a pair of left and right braces as in lines 6 and 20 of Fig. 3.1.

Identifiers and Camel Case Naming

Class names, method names and variable names are all *identifiers* and by convention all use the same *camel case* naming scheme we discussed in Chapter 2. Also by convention, class

names begin with an initial *uppercase* letter, and method names and variable names begin with an initial *lowercase* letter.

Instance Variable name

Recall that an object has attributes, implemented as instance variables and carried with it throughout its lifetime. Instance variables exist before methods are called on an object, while the methods are executing and after the methods complete execution. Each object (instance) of the class has its *own* copy of the class's instance variables. A class normally contains one or more methods that manipulate the instance variables belonging to particular objects of the class.

Instance variables are declared *inside* a class declaration but *outside* the bodies of the class's methods. Line 7

```
private String name; // instance variable
```

declares instance variable `name` of type `String` *outside* the bodies of methods `setName` (lines 10–13) and `getName` (lines 16–19). `String` variables can hold character string values such as "Jane Green". If there are many `Account` objects, each has its own `name`. Because `name` is an instance variable, it can be manipulated by each of the class's methods.



Good Programming Practice 3.1

We prefer to list a class's instance variables first in the class's body, so that you see the names and types of the variables before they're used in the class's methods. You can list the class's instance variables anywhere in the class outside its method declarations, but scattering the instance variables can lead to hard-to-read code.

Access Modifiers `public` and `private`

Most instance-variable declarations are preceded with the keyword `private` (as in line 7). Like `public`, `private` is an *access modifier*. Variables or methods declared with access modifier `private` are accessible only to methods of the class in which they're declared. So, the variable `name` can be used only in each `Account` object's methods (`setName` and `getName` in this case). You'll soon see that this presents powerful software engineering opportunities.

setName Method of Class Account

Let's walk through the code of `setName`'s method declaration (lines 10–13):

```
public void setName(String name) — This line is the method header
{
    this.name = name; // store the name
}
```

We refer to the first line of each method declaration (line 10 in this case) as the *method header*. The method's return type (which appears before the method name) specifies the type of data the method returns to its caller after performing its task. The return type `void` (line 10) indicates that `setName` will perform a task but will not return (i.e., give back) any information to its caller. In Chapter 2, you used methods that return information—for example, you used `Scanner` method `nextInt` to input an integer typed by the user at the keyboard. When `nextInt` reads a value from the user, it returns that value for use in the program. As you'll soon see, `Account` method `getName` returns a value.

Method `setName` receives parameter `name` of type `String`. Parameters are declared in the parameter list, which is located inside the parentheses that follow the method name in the method header. When there are multiple parameters, each is separated from the next by a comma. Each parameter must specify a type (in this case, `String`) followed by a variable name (in this case, `name`).

Parameters Are Local Variables

In Chapter 2, we declared all of an app’s variables in the `main` method. Variables declared in a particular method’s body (such as `main`) are local variables which can be used only in that method. Each method can access only its own local variables, not those of other methods. When a method terminates, the values of its local variables are lost. A method’s parameters also are local variables of the method.

setName Method Body

Every method body is delimited by a pair of braces (as in lines 11 and 13 of Fig. 3.1) containing one or more statements that perform the method’s task(s). In this case, the method body contains a single statement (line 12) that assigns the value of the `name` parameter (a `String`) to the class’s `name` instance variable, thus storing the account name in the object.

If a method contains a local variable with the same name as an instance variable (as in lines 10 and 7, respectively), that method’s body will refer to the local variable rather than the instance variable. In this case, the local variable is said to *shadow* the instance variable in the method’s body. The method’s body can use the keyword `this` to refer to the shadowed instance variable explicitly, as shown on the left side of the assignment in line 12.



Good Programming Practice 3.2

We could have avoided the need for keyword `this` here by choosing a different name for the parameter in line 10, but using the `this` keyword as shown in line 12 is a widely accepted practice to minimize the proliferation of identifier names.

After line 12 executes, the method has completed its task, so it returns to its *caller*. As you’ll soon see, the statement in line 21 of `main` (Fig. 3.2) calls method `setName`.

getName Method of Class Account

Method `getName` (lines 16–19 of Fig. 3.1)

```
public String getName()
{
    return name; // return value of name to caller
}
```

Keyword `return` passes the `String` `name` back to the method’s caller

returns a particular `Account` object’s `name` to the caller. The method has an empty parameter list, so it does not require additional information to perform its task. The method returns a `String`. When a method that specifies a return type other than `void` is called and completes its task, it must return a result to its caller. A statement that calls method `getName` on an `Account` object (such as the ones in lines 16 and 26 of Fig. 3.2) expects to receive the `Account`’s `name`—a `String`, as specified in the method declaration’s return type.

The return statement in line 18 of Fig. 3.1 passes the `String` value of instance variable `name` back to the caller. For example, when the value is returned to the statement in lines 25–26 of Fig. 3.2, the statement uses that value to output the name.

3.2.2 AccountTest Class That Creates and Uses an Object of Class Account

Next, we'd like to use class `Account` in an app and *call* each of its methods. A class that contains a `main` method begins the execution of a Java app. Class `Account` cannot execute by itself because it does not contain a `main` method—if you type `java Account` in the command window, you'll get an error indicating “Main method not found in class `Account`.¹” To fix this problem, you must either declare a separate class that contains a `main` method or place a `main` method in class `Account`.

Driver Class AccountTest

We use a separate class `AccountTest` (Fig. 3.2) containing method `main` to test class `Account`. Once `main` begins executing, it may call other methods in this and other classes; those may, in turn, call other methods, and so on. Class `AccountTest`'s `main` method creates one `Account` object and calls its `getName` and `setName` methods. Such a class is sometimes called a *driver class*—just as a `Person` object drives a `Car` object by telling it what to do (go faster, go slower, turn left, turn right, etc.), class `AccountTest` drives an `Account` object, telling it what to do by calling its methods.

```

1 // Fig. 3.2: AccountTest.java
2 // Creating and manipulating an Account object.
3 import java.util.Scanner;
4
5 public class AccountTest
6 {
7     public static void main(String[] args)
8     {
9         // create a Scanner object to obtain input from the command window
10        Scanner input = new Scanner(System.in);
11
12        // create an Account object and assign it to myAccount
13        Account myAccount = new Account();
14
15        // display initial value of name (null)
16        System.out.printf("Initial name is: %s%n%n",
17                         myAccount.getName());
18
19        // prompt for and read name
20        System.out.println("Please enter the name:");
21        String theName = input.nextLine(); // read a line of text
22        myAccount.setName(theName); // put theName in myAccount
23        System.out.println(); // outputs a blank line
24
25        // display the name stored in object myAccount
26        System.out.printf("Name in object myAccount is:%n%s%n",
27                         myAccount.getName());
28    }
29 } // end class AccountTest

```

Fig. 3.2 | Creating and manipulating an `Account` object. (Part 1 of 2.)

```

Initial name is: null
Please enter the name:
Jane Green
Name in object myAccount is:
Jane Green

```

Fig. 3.2 | Creating and manipulating an Account object. (Part 2 of 2.)

Scanner Object for Receiving Input from the User

Line 10 creates a Scanner object named `input` for inputting the name from the user. Line 19 prompts the user to enter a name. Line 20 uses the Scanner object's `nextLine` method to read the name from the user and assign it to the local variable `theName`. You type the name and press *Enter* to submit it to the program. Pressing *Enter* inserts a newline character after the characters you typed. Method `nextLine` reads characters (including white-space characters, such as the blank in "Jane Green") until it encounters the newline, then returns a `String` containing the characters up to, but *not* including, the newline, which is discarded.

Class `Scanner` provides various other input methods, as you'll see throughout the book. A method similar to `nextLine`—named `next`—reads the next word. When you press *Enter* after typing some text, method `next` reads characters until it encounters a white-space character (such as a space, tab or newline), then returns a `String` containing the characters up to, but *not* including, the white-space character, which is discarded. All information after the first white-space character is not lost—it can be read by subsequent statements that call the `Scanner`'s methods later in the program.

Instantiating an Object—Keyword `new` and Constructors

Line 13 creates an `Account` object and assigns it to variable `myAccount` of type `Account`. Variable `myAccount` is initialized with the result of the class instance creation expression `new Account()`. Keyword `new` creates a new object of the specified class—in this case, `Account`. The parentheses to the right of `Account` are required. As you'll learn in Section 3.4, those parentheses in combination with a class name represent a call to a **constructor**, which is similar to a method but is called implicitly by the `new` operator to initialize an object's instance variables when the object is created. In Section 3.4, you'll see how to place an argument in the parentheses to specify an initial value for an `Account` object's `name` instance variable—you'll enhance class `Account` to enable this. For now, we simply leave the parentheses empty. Line 10 contains a class instance creation expression for a `Scanner` object—the expression initializes the `Scanner` with `System.in`, which tells the `Scanner` where to read the input from (i.e., the keyboard).

Calling Class `Account`'s `getName` Method

Line 16 displays the initial name, which is obtained by calling the object's `getName` method. Just as we can use object `System.out` to call its methods `print`, `printf` and `println`, we can use object `myAccount` to call its methods `getName` and `setName`. Line 16 calls `getName` using the `myAccount` object created in line 13, followed by a dot separator `(.)`,

then the method name `getName` and an empty set of parentheses because no arguments are being passed. When `getName` is called:

1. The app transfers program execution from the call (line 16 in `main`) to method `getName`'s declaration (lines 16–19 of Fig. 3.1). Because `getName` was called via the `myAccount` object, `getName` “knows” which object's instance variable to manipulate.
2. Next, method `getName` performs its task—that is, it returns the name (line 18 of Fig. 3.1). When the `return` statement executes, program execution continues where `getName` was called (line 16 in Fig. 3.2).
3. `System.out.printf` displays the `String` returned by method `getName`, then the program continues executing at line 19 in `main`.



Error-Prevention Tip 3.1

Never use as a format-control a string that was input from the user. When method `System.out.printf` evaluates the format-control string in its first argument, the method performs tasks based on the conversion specifier(s) in that string. If the format-control string were obtained from the user, a malicious user could supply conversion specifiers that would be executed by `System.out.printf`, possibly causing a security breach.

`null`—the Default Initial Value for `String` Variables

The first line of the output shows the name “`null`.¹ Unlike local variables, which are *not* automatically initialized, every instance variable has a default initial value—a value provided by Java when you do not specify the instance variable's initial value. Thus, instance variables are not required to be explicitly initialized before they're used in a program—unless they must be initialized to values other than their default values. The default value for an instance variable of type `String` (like `name` in this example) is `null`, which we discuss further in Section 3.3 when we consider reference types.

Calling Class `Account`'s `setName` Method

Line 21 calls `myAccounts`'s `setName` method. A method call can supply arguments whose values are assigned to the corresponding method parameters. In this case, the value of `main`'s local variable `theName` in parentheses is the argument that's passed to `setName` so that the method can perform its task. When `setName` is called:

1. The app transfers program execution from line 21 in `main` to `setName` method's declaration (lines 10–13 of Fig. 3.1), and the argument value in the call's parentheses (`theName`) is assigned to the corresponding parameter (`name`) in the method header (line 10 of Fig. 3.1). Because `setName` was called via the `myAccount` object, `setName` “knows” which object's instance variable to manipulate.
2. Next, method `setName` performs its task—that is, it assigns the `name` parameter's value to instance variable `name` (line 12 of Fig. 3.1).
3. When program execution reaches `setName`'s closing right brace, it returns to where `setName` was called (line 21 of Fig. 3.2), then continues at line 22.

The number of arguments in a method call must match the number of parameters in the method declaration's parameter list. Also, the argument types in the method call must be consistent with the types of the corresponding parameters in the method's declaration. (As you'll see in Chapter 6, an argument's type and its corresponding parameter's type are

(*not required to be identical.*) In our example, the method call passes one argument of type `String` (`theName`)—and the method declaration specifies one parameter of type `String` (`name`, declared in line 10 of Fig. 3.1). So in this example, the type of the argument in the method call exactly matches the type of the parameter in the method header.

Displaying the Name That Was Entered by the User

Line 22 of Fig. 3.2 outputs a blank line. When the second call to method `getName` (line 26) executes, the name entered by the user in line 20 is displayed. When the statement at lines 25–26 completes execution, the end of method `main` is reached, so the program terminates.

3.2.3 Compiling and Executing an App with Multiple Classes

You must compile the classes in Figs. 3.1 and 3.2 before you can execute the app. This is the first time you've created an app with multiple classes. Class `AccountTest` has a `main` method; class `Account` does not. To compile this app, first change to the directory that contains the app's source-code files. Next, type the command

```
javac Account.java AccountTest.java
```

to compile both classes at once. If the directory containing the app includes *only* this app's files, you can compile both classes with the command

```
javac *.java
```

The asterisk (*) in `*.java` indicates that all files in the current directory ending with the filename extension ".java" should be compiled. If both classes compile correctly—that is, no compilation errors are displayed—you can then run the app with the command

```
java AccountTest
```

3.2.4 Account UML Class Diagram with an Instance Variable and set and get Methods

We'll often use UML class diagrams to summarize a class's attributes and operations. In industry, UML diagrams help systems designers specify a system in a concise, graphical, programming-language-independent manner, before programmers implement the system in a specific programming language. Figure 3.3 presents a UML class diagram for class `Account` of Fig. 3.1.

Top Compartment

In the UML, each class is modeled in a class diagram as a rectangle with three compartments. In this diagram the top compartment contains the class name `Account` centered horizontally in boldface type.

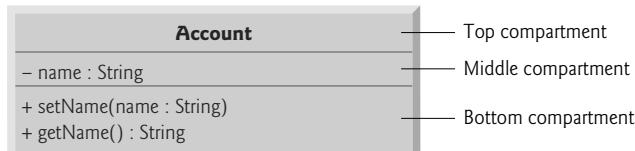


Fig. 3.3 | UML class diagram for class `Account` of Fig. 3.1.

Middle Compartment

The middle compartment contains the class's attribute name, which corresponds to the instance variable of the same name in Java. Instance variable `name` is `private` in Java, so the UML class diagram lists a minus sign (–) access modifier before the attribute name. Following the attribute name are a colon and the attribute type, in this case `String`.

Bottom Compartment

The bottom compartment contains the class's `operations`, `setName` and `getName`, which correspond to the methods of the same names in Java. The UML models operations by listing the operation name preceded by an access modifier, in this case `+ getName`. This plus sign (+) indicates that `getName` is a public operation in the UML (because it's a `public` method in Java). Operation `getName` does not have any parameters, so the parentheses following the operation name in the class diagram are empty, just as they are in the method's declaration in line 16 of Fig. 3.1. Operation `setName`, also a public operation, has a `String` parameter called `name`.

Return Types

The UML indicates the return type of an operation by placing a colon and the return type after the parentheses following the operation name. Account method `getName` (Fig. 3.1) has a `String` return type. Method `setName` does not return a value (because it returns `void` in Java), so the UML class diagram does not specify a return type after the parentheses of this operation.

Parameters

The UML models a parameter a bit differently from Java by listing the parameter name, followed by a colon and the parameter type in the parentheses after the operation name. The UML has its own data types similar to those of Java, but for simplicity, we'll use the Java data types. Account method `setName` (Fig. 3.1) has a `String` parameter named `name`, so Fig. 3.3 lists `name : String` between the parentheses following the method name.

3.2.5 Additional Notes on Class `AccountTest`

`static` Method `main`

In Chapter 2, each class we declared had one method named `main`. Recall that `main` is a special method that's always called automatically by the Java Virtual Machine (JVM) when you execute an app. You must call most other methods explicitly to tell them to perform their tasks.

Lines 7–27 of Fig. 3.2 declare method `main`. A key part of enabling the JVM to locate and call method `main` to begin the app's execution is the `static` keyword (line 7), which indicates that `main` is a `static` method. A `static` method is special, because you can call it *without first creating an object of the class in which the method is declared*—in this case class `AccountTest`. We discuss `static` methods in detail in Chapter 6.

Notes on import Declarations

Notice the `import` declaration in Fig. 3.2 (line 3), which indicates to the compiler that the program uses class `Scanner`. As mentioned in Chapter 2, classes `System` and `String` are in

package `java.lang`, which is *implicitly* imported into every Java program, so all programs can use that package's classes without explicitly importing them. Most other classes you'll use in Java programs must be imported explicitly.

There's a special relationship between classes that are compiled in the same directory, like classes `Account` and `AccountTest`. By default, such classes are considered to be in the same package—known as the **default package**. Classes in the same package are implicitly imported into the source-code files of other classes in that package. Thus, an `import` declaration is not required when one class in a package uses another in the same package—such as when class `AccountTest` uses class `Account`.

The `import` declaration in line 3 is *not* required if we refer to class `Scanner` throughout this file as `java.util.Scanner`, which includes the full package name and class name. This is known as the class's **fully qualified class name**. For example, line 10 of Fig. 3.2 also could be written as

```
java.util.Scanner input = new java.util.Scanner(System.in);
```



Software Engineering Observation 3.1

The Java compiler does not require import declarations in a Java source-code file if the fully qualified class name is specified every time a class name is used. Most Java programmers prefer the more concise programming style enabled by import declarations.

3.2.6 Software Engineering with `private` Instance Variables and `public` `set` and `get` Methods

As you'll see, through the use of `set` and `get` methods, you can *validate* attempted modifications to `private` data and control how that data is presented to the caller—these are compelling software engineering benefits. We'll discuss this in more detail in Section 3.5.

If the instance variable were `public`, any client of the class—that is, any other class that calls the class's methods—could see the data and do whatever it wanted with it, including setting it to an invalid value.

You might think that even though a client of the class cannot directly access a `private` instance variable, the client can do whatever it wants with the variable through `public` `set` and `get` methods. You would think that you could peek at the `private` data any time with the `public` `get` method and that you could modify the `private` data at will through the `public` `set` method. But `set` methods can be programmed to validate their arguments and reject any attempts to `set` the data to bad values, such as a negative body temperature, a day in March out of the range 1 through 31, a product code not in the company's product catalog, etc. And a `get` method can present the data in a different form. For example, a `Grade` class might store a grade as an `int` between 0 and 100, but a `getGrade` method might return a letter grade as a `String`, such as "A" for grades between 90 and 100, "B" for grades between 80 and 89, etc. Tightly controlling the access to and presentation of `private` data can greatly reduce errors, while increasing the robustness and security of your programs.

Declaring instance variables with access modifier `private` is known as *data hiding* or *information hiding*. When a program creates (instantiates) an object of class `Account`, variable `name` is *encapsulated* (hidden) in the object and can be accessed only by methods of the object's class.



Software Engineering Observation 3.2

Precede each instance variable and method declaration with an access modifier. Generally, instance variables should be declared private and methods public. Later in the book, we'll discuss why you might want to declare a method private.

Conceptual View of an Account Object with Encapsulated Data

You can think of an Account object as shown in Fig. 3.4. The private instance variable name is *hidden* inside the object (represented by the inner circle containing name) and *protected* by an outer layer of public methods (represented by the outer circle containing getName and setName). Any client code that needs to interact with the Account object can do so *only* by calling the public methods of the protective outer layer.

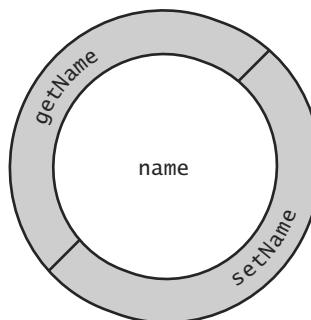


Fig. 3.4 | Conceptual view of an Account object with its encapsulated private instance variable name and protective layer of public methods.

3.3 Primitive Types vs. Reference Types

Java's types are divided into primitive types and reference types. In Chapter 2, you worked with variables of type `int`—one of the primitive types. The other primitive types are `boolean`, `byte`, `char`, `short`, `long`, `float` and `double`, each of which we discuss in this book—these are summarized in Appendix D. All nonprimitive types are reference types, so classes, which specify the types of objects, are reference types.

A primitive-type variable can hold exactly *one* value of its declared type at a time. For example, an `int` variable can store one integer at a time. When another value is assigned to that variable, the new value replaces the previous one—which is *lost*.

Recall that local variables are *not* initialized by default. Primitive-type instance variables *are* initialized by default—instance variables of types `byte`, `char`, `short`, `int`, `long`, `float` and `double` are initialized to 0, and variables of type `boolean` are initialized to `false`. You can specify your own initial value for a primitive-type variable by assigning the variable a value in its declaration, as in

```
private int numberOfStudents = 10;
```

Programs use variables of reference types (normally called references) to store the *addresses* of objects in the computer's memory. Such a variable is said to refer to an object

in the program. *Objects* that are referenced may each contain *many* instance variables. Line 10 of Fig. 3.2:

```
Scanner input = new Scanner(System.in);
```

creates an object of class `Scanner`, then assigns to the variable `input` a *reference* to that `Scanner` object. Line 13 of Fig. 3.2:

```
Account myAccount = new Account();
```

creates an object of class `Account`, then assigns to the variable `myAccount` a *reference* to that `Account` object. *Reference-type instance variables, if not explicitly initialized, are initialized by default to the value null*—which represents a “reference to nothing.” That’s why the first call to `getName` in line 16 of Fig. 3.2 returns `null`—the value of `name` has *not* yet been set, so the *default initial value null* is returned.

To call methods on an object, you need a reference to the object. In Fig. 3.2, the statements in method `main` use the variable `myAccount` to call methods `getName` (lines 16 and 26) and `setName` (line 21) to interact with the `Account` object. Primitive-type variables do *not* refer to objects, so such variables *cannot* be used to call methods.

3.4 Account Class: Initializing Objects with Constructors

As mentioned in Section 3.2, when an object of class `Account` (Fig. 3.1) is created, its `String` instance variable `name` is initialized to `null` by *default*. But what if you want to provide a name when you *create* an `Account` object?

Each class you declare can optionally provide a *constructor* with parameters that can be used to initialize an object of a class when the object is created. Java *requires* a constructor call for *every* object that’s created, so this is the ideal point to initialize an object’s instance variables. The next example enhances class `Account` (Fig. 3.5) with a constructor that can receive a name and use it to initialize instance variable `name` when an `Account` object is created (Fig. 3.6).

3.4.1 Declaring an Account Constructor for Custom Object Initialization

When you declare a class, you can provide your own constructor to specify *custom initialization* for objects of your class. For example, you might want to specify a name for an `Account` object when the object is created, as in line 10 of Fig. 3.6:

```
Account account1 = new Account("Jane Green");
```

In this case, the `String` argument "Jane Green" is passed to the `Account` object’s constructor and used to initialize the `name` instance variable. The preceding statement requires that the class provide a constructor that takes only a `String` parameter. Figure 3.5 contains a modified `Account` class with such a constructor.

```
1 // Fig. 3.5: Account.java
2 // Account class with a constructor that initializes the name.
3
```

Fig. 3.5 | `Account` class with a constructor that initializes the `name`. (Part 1 of 2.)

```
4 public class Account
5 {
6     private String name; // instance variable
7
8     // constructor initializes name with parameter name
9     public Account(String name) // constructor name is class name
10    {
11        this.name = name;
12    }
13
14    // method to set the name
15    public void setName(String name)
16    {
17        this.name = name;
18    }
19
20    // method to retrieve the name
21    public String getName()
22    {
23        return name;
24    }
25 } // end class Account
```

Fig. 3.5 | Account class with a constructor that initializes the name. (Part 2 of 2.)

Account Constructor Declaration

Lines 9–12 of Fig. 3.5 declare Account’s constructor. A constructor *must* have the *same name* as the class. A constructor’s *parameter list* specifies that the constructor requires one or more pieces of data to perform its task. Line 9 indicates that the constructor has a *String* parameter called *name*. When you create a new *Account* object (as you’ll see in Fig. 3.6), you’ll pass a person’s name to the constructor, which will receive that name in the parameter *name*. The constructor will then assign *name* to *instance variable* *name* in line 11.



Error-Prevention Tip 3.2

Even though it’s possible to do so, do not call methods from constructors. We’ll explain this in Chapter 10, Object-Oriented Programming: Polymorphism and Interfaces.

Parameter name of Class Account’s Constructor and Method setName

Recall from Section 3.2.1 that method parameters are local variables. In Fig. 3.5, the constructor and method *setName* both have a parameter called *name*. Although these parameters have the same identifier (*name*), the parameter in line 9 is a local variable of the constructor that’s *not* visible to method *setName*, and the one in line 15 is a local variable of *setName* that’s *not* visible to the constructor.

3.4.2 Class AccountTest: Initializing Account Objects When They’re Created

The *AccountTest* program (Fig. 3.6) initializes two *Account* objects using the constructor. Line 10 creates and initializes the *Account* object *account1*. Keyword *new* requests memory from the system to store the *Account* object, then implicitly calls the class’s con-

structor to *initialize* the object. The call is indicated by the parentheses after the class name, which contain the *argument* "Jane Green" that's used to initialize the new object's name. The class instance creation expression in line 10 returns a *reference* to the new object, which is assigned to the variable account1. Line 11 repeats this process, passing the argument "John Blue" to initialize the name for account2. Lines 14–15 use each object's getName method to obtain the names and show that they were indeed initialized when the objects were *created*. The output shows *different* names, confirming that each Account maintains its *own copy* of instance variable name.

```

1 // Fig. 3.6: AccountTest.java
2 // Using the Account constructor to initialize the name instance
3 // variable at the time each Account object is created.
4
5 public class AccountTest
6 {
7     public static void main(String[] args)
8     {
9         // create two Account objects
10        Account account1 = new Account("Jane Green");
11        Account account2 = new Account("John Blue");
12
13        // display initial value of name for each Account
14        System.out.printf("account1 name is: %s%n", account1.getName());
15        System.out.printf("account2 name is: %s%n", account2.getName());
16    }
17 } // end class AccountTest

```

```
account1 name is: Jane Green
account2 name is: John Blue
```

Fig. 3.6 | Using the Account constructor to initialize the name instance variable at the time each Account object is created.

Constructors Cannot Return Values

An important difference between constructors and methods is that *constructors cannot return values*, so they *cannot* specify a return type (not even void). Normally, constructors are declared **public**—later in the book we'll explain when to use **private** constructors.

Default Constructor

Recall that line 13 of Fig. 3.2

```
Account myAccount = new Account();
```

used **new** to create an Account object. The *empty* parentheses after "new Account" indicate a call to the class's **default constructor**—in any class that does *not* explicitly declare a constructor, the compiler provides a default constructor (which always has no parameters). When a class has only the default constructor, the class's instance variables are initialized to their *default values*. In Section 8.5, you'll learn that classes can have multiple constructors.

There's No Default Constructor in a Class That Declares a Constructor

If you declare a constructor for a class, the compiler will *not* create a *default constructor* for that class. In that case, you will not be able to create an Account object with the class instance creation expression `new Account()` as we did in Fig. 3.2—unless the custom constructor you declare takes *no* parameters.



Software Engineering Observation 3.3

Unless default initialization of your class's instance variables is acceptable, provide a custom constructor to ensure that your instance variables are properly initialized with meaningful values when each new object of your class is created.

Adding the Constructor to Class Account's UML Class Diagram

The UML class diagram of Fig. 3.7 models class Account of Fig. 3.5, which has a constructor with a String name parameter. Like operations, the UML models constructors in the *third* compartment of a class diagram. To distinguish a constructor from the class's operations, the UML requires that the word “constructor” be enclosed in guillemets (« and ») and placed before the constructor’s name. It’s customary to list constructors *before* other operations in the third compartment.

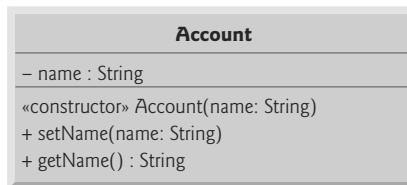


Fig. 3.7 | UML class diagram for Account class of Fig. 3.5.

3.5 Account Class with a Balance; Floating-Point Numbers

We now declare an Account class that maintains the *balance* of a bank account in addition to the name. Most account balances are not integers. So, class Account represents the account balance as a **floating-point number**—a number with a *decimal point*, such as 43.95, 0.0, –129.8873. [In Chapter 8, we’ll begin representing monetary amounts precisely with class `BigDecimal` as you should do when writing industrial-strength monetary applications.]

Java provides two primitive types for storing floating-point numbers in memory—`float` and `double`. Variables of type `float` represent single-precision floating-point numbers and can hold up to *seven significant digits*. Variables of type `double` represent double-precision floating-point numbers. These require *twice* as much memory as `float` variables and can hold up to *15 significant digits*—about *double* the precision of `float` variables.

Most programmers represent floating-point numbers with type `double`. In fact, Java treats all floating-point numbers you type in a program’s source code (such as 7.33 and 0.0975) as `double` values by default. Such values in the source code are known as **floating-point literals**. See Appendix D, Primitive Types, for the precise ranges of values for `floats` and `doubles`.

3.5.1 Account Class with a balance Instance Variable of Type double

Our next app contains a version of class `Account` (Fig. 3.8) that maintains as instance variables the `name` and the `balance` of a bank account. A typical bank services *many* accounts, each with its *own* balance, so line 8 declares an instance variable `balance` of type `double`. Every instance (i.e., object) of class `Account` contains its *own* copies of *both* the `name` and the `balance`.

```
1 // Fig. 3.8: Account.java
2 // Account class with a double instance variable balance and a constructor
3 // and deposit method that perform validation.
4
5 public class Account
6 {
7     private String name; // instance variable
8     private double balance; // instance variable
9
10    // Account constructor that receives two parameters
11    public Account(String name, double balance)
12    {
13        this.name = name; // assign name to instance variable name
14
15        // validate that the balance is greater than 0.0; if it's not,
16        // instance variable balance keeps its default initial value of 0.0
17        if (balance > 0.0) // if the balance is valid
18            this.balance = balance; // assign it to instance variable balance
19    }
20
21    // method that deposits (adds) only a valid amount to the balance
22    public void deposit(double depositAmount)
23    {
24        if (depositAmount > 0.0) // if the depositAmount is valid
25            balance = balance + depositAmount; // add it to the balance
26    }
27
28    // method returns the account balance
29    public double getBalance()
30    {
31        return balance;
32    }
33
34    // method that sets the name
35    public void setName(String name)
36    {
37        this.name = name;
38    }
39
40    // method that returns the name
41    public String getName()
42    {
```

Fig. 3.8 | Account class with a `double` instance variable `balance` and a constructor and `deposit` method that perform validation. (Part I of 2.)

```
43     return name; // give value of name back to caller
44   } // end method getName
45 } // end class Account
```

Fig. 3.8 | Account class with a `double` instance variable `balance` and a constructor and `deposit` method that perform validation. (Part 2 of 2.)

Account Class Two-Parameter Constructor

The class has a *constructor* and four *methods*. It's common for someone opening an account to deposit money immediately, so the constructor (lines 11–19) now receives a second parameter—`initialBalance` of type `double` that represents the *starting balance*. Lines 17–18 ensure that `initialBalance` is greater than `0.0`. If so, `initialBalance`'s value is assigned to instance variable `balance`. Otherwise, `balance` remains at `0.0`—its *default initial value*.

Account Class `deposit` Method

Method `deposit` (lines 22–26) does *not* return any data when it completes its task, so its return type is `void`. The method receives one parameter named `depositAmount`—a `double` value that's *added* to the `balance` *only* if the parameter value is *valid* (i.e., greater than zero). Line 25 first adds the current `balance` and `depositAmount`, forming a *temporary* sum which is *then* assigned to `balance`, *replacing* its prior value (recall that addition has a *higher* precedence than assignment). It's important to understand that the calculation on the right side of the assignment operator in line 25 does *not* modify the `balance`—that's why the assignment is necessary.

Account Class `getBalance` Method

Method `getBalance` (lines 29–32) allows *clients* of the class (i.e., other classes whose methods call the methods of this class) to obtain the value of a particular `Account` object's `balance`. The method specifies return type `double` and an *empty* parameter list.

Account's Methods Can All Use `balance`

Once again, the statements in lines 18, 25 and 31 use the variable `balance` even though it was *not* declared in *any* of the methods. We can use `balance` in these methods because it's an *instance variable* of the class.

3.5.2 AccountTest Class to Use Class Account

Class `AccountTest` (Fig. 3.9) creates two `Account` objects (lines 9–10) and initializes them with a *valid* balance of `50.00` and an *invalid* balance of `-7.53`, respectively—for the purpose of our examples, we assume that balances must be greater than or equal to zero. The calls to method `System.out.printf` in lines 13–16 output the account names and balances, which are obtained by calling each `Account`'s `getName` and `getBalance` methods.

```
1 // Fig. 3.9: AccountTest.java
2 // Inputting and outputting floating-point numbers with Account objects.
3 import java.util.Scanner;
```

Fig. 3.9 | Inputting and outputting floating-point numbers with `Account` objects. (Part 1 of 3.)

```
4  public class AccountTest
5  {
6      public static void main(String[] args)
7      {
8          Account account1 = new Account("Jane Green", 50.00);
9          Account account2 = new Account("John Blue", -7.53);
10
11         // display initial balance of each object
12         System.out.printf("%s balance: $%.2f%n",
13                           account1.getName(), account1.getBalance());
14         System.out.printf("%s balance: $%.2f%n%n",
15                           account2.getName(), account2.getBalance());
16
17         // create a Scanner to obtain input from the command window
18         Scanner input = new Scanner(System.in);
19
20         System.out.print("Enter deposit amount for account1: "); // prompt
21         double depositAmount = input.nextDouble(); // obtain user input
22         System.out.printf("%nadding %.2f to account1 balance%n%n",
23                           depositAmount);
24         account1.deposit(depositAmount); // add to account1's balance
25
26         // display balances
27         System.out.printf("%s balance: $%.2f%n",
28                           account1.getName(), account1.getBalance());
29         System.out.printf("%s balance: $%.2f%n%n",
30                           account2.getName(), account2.getBalance());
31
32         System.out.print("Enter deposit amount for account2: "); // prompt
33         depositAmount = input.nextDouble(); // obtain user input
34         System.out.printf("%nadding %.2f to account2 balance%n%n",
35                           depositAmount);
36         account2.deposit(depositAmount); // add to account2 balance
37
38         // display balances
39         System.out.printf("%s balance: $%.2f%n",
40                           account1.getName(), account1.getBalance());
41         System.out.printf("%s balance: $%.2f%n%n",
42                           account2.getName(), account2.getBalance());
43     } // end main
44 } // end class AccountTest
```

```
Jane Green balance: $50.00
John Blue balance: $0.00

Enter deposit amount for account1: 25.53

adding 25.53 to account1 balance

Jane Green balance: $75.53
John Blue balance: $0.00
```

Fig. 3.9 | Inputting and outputting floating-point numbers with `Account` objects. (Part 2 of 3.)

```
Enter deposit amount for account2: 123.45
adding 123.45 to account2 balance
Jane Green balance: $75.53
John Blue balance: $123.45
```

Fig. 3.9 | Inputting and outputting floating-point numbers with `Account` objects. (Part 3 of 3.)

Displaying the `Account` Objects' Initial Balances

When method `getBalance` is called for `account1` from line 14, the value of `account1`'s balance is returned from line 31 of Fig. 3.8 and displayed by the `System.out.printf` statement (Fig. 3.9, lines 13–14). Similarly, when method `getBalance` is called for `account2` from line 16, the value of the `account2`'s balance is returned from line 31 of Fig. 3.8 and displayed by the `System.out.printf` statement (Fig. 3.9, lines 15–16). The balance of `account2` is initially 0.00, because the constructor rejected the attempt to start `account2` with a *negative* balance, so the balance retains its default initial value.

Formatting Floating-Point Numbers for Display

Each of the balances is output by `printf` with the format specifier `.2f`. The `%f` format specifier is used to output values of type `float` or `double`. The `.2` between `%` and `f` represents the number of decimal places (2) that should be output to the right of the decimal point in the floating-point number—also known as the number's precision. Any floating-point value output with `.2f` will be rounded to the hundredths *position*—for example, 123.457 would be rounded to 123.46 and 27.33379 would be rounded to 27.33.

Reading a Floating-Point Value from the User and Making a Deposit

Line 21 (Fig. 3.9) prompts the user to enter a deposit amount for `account1`. Line 22 declares local variable `depositAmount` to store each deposit amount entered by the user. Unlike instance variables (such as `name` and `balance` in class `Account`), local variables (like `depositAmount` in `main`) are not initialized by default, so they normally must be initialized explicitly. As you'll learn momentarily, variable `depositAmount`'s initial value will be determined by the user's input.



Common Programming Error 3.1

The Java compiler will issue a compilation error if you attempt to use the value of an uninitialized local variable. This helps you avoid dangerous execution-time logic errors. It's always better to get the errors out of your programs at compilation time rather than execution time.

Line 22 obtains the input from the user by calling `Scanner` object `input`'s `nextDouble` method, which returns a `double` value entered by the user. Lines 23–24 display the `depositAmount`. Line 25 calls object `account1`'s `deposit` method with the `depositAmount` as the method's argument. When the method is called, the argument's value is assigned to the parameter `depositAmount` of method `deposit` (line 22 of Fig. 3.8); then method `deposit` adds that value to the `balance`. Lines 28–31 (Fig. 3.9) output the names and balances of both `Accounts` again to show that *only* `account1`'s `balance` has changed.

Line 33 prompts the user to enter a deposit amount for account2. Line 34 obtains the input from the user by calling Scanner object input's nextDouble method. Lines 35–36 display the depositAmount. Line 37 calls object account2's deposit method with depositAmount as the method's argument; then method deposit adds that value to the balance. Finally, lines 40–43 output the names and balances of both Accounts again to show that only account2's balance has changed.

UML Class Diagram for Class Account

The UML class diagram in Fig. 3.10 concisely models class Account of Fig. 3.8. The diagram models in its second compartment the private attributes name of type String and balance of type double.

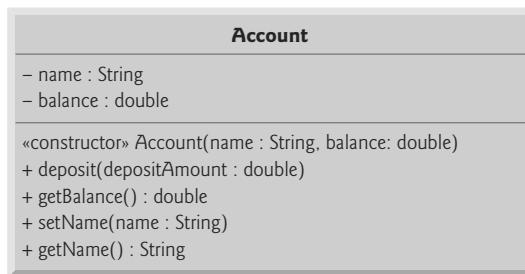


Fig. 3.10 | UML class diagram for Account class of Fig. 3.8.

Class Account's constructor is modeled in the third compartment with parameters name of type String and initialBalance of type double. The class's four public methods also are modeled in the *third* compartment—operation deposit with a depositAmount parameter of type double, operation getBalance with a return type of double, operation setName with a name parameter of type String and operation getName with a return type of String.

3.6 Wrap-Up

In this chapter, you learned how to create your own Java classes and methods, create objects of those classes and call methods of those objects to perform useful actions. You declared instance variables of a class to maintain data for each object of the class, and you declared your own methods to operate on that data. You called methods and passed information to them as arguments whose values are assigned to the method's parameters. You learned the difference between a local variable of a method and an instance variable of a class, and that only instance variables are initialized automatically. You also learned how to use a class's constructor to specify the initial values for an object's instance variables. You saw how to create UML class diagrams that model the methods, attributes and constructors of classes. Finally, you learned about floating-point numbers (numbers with decimal points)—how to store them with variables of primitive type double, how to input them with a Scanner object and how to format them with printf and format specifier %f for display purposes. [In Chapter 8, we'll begin representing monetary amounts precisely with class BigDecimal.] In the next chapter we discuss control statements, which specify the order in which a program's actions are performed.

This page intentionally left blank

Index

Symbols

`&`, bitwise exclusive OR 973
`&`, boolean logical exclusive OR 110, 112
 truth table 112
`&=`, bitwise exclusive OR assignment operator 983
`-` SQL wildcard character 740, 741
`,` (comma) formatting flag 96
`--`, predecrement/postdecrement 82
`-`, subtraction 33, 34
`!`, logical NOT 110, 112
 truth table 113
`!=`, not equals 34
`? (wildcard type argument)` 607
`? :` ternary conditional operator 66
`.` dot separator 44
`, flag` 956
`(flag` 956
`{, left brace` 23
`}, right brace` 23
`@ symbol` 924
`@Override annotation` 243
`* SQL wildcard character` 739
`* wildcard in a file name` 46
`*, multiplication 33, 34
*=, multiplication assignment operator 81
/, division 33, 34
/* */ traditional comment 22
/** */ Java documentation comment 22, 924
//, end-of-line comment 22
/=, division assignment operator 81
\, backslash escape sequence 28
\', single-quote-character escape sequence 960
\", double-quote escape sequence 28, 960
\\\, backslash-character escape sequence 960
\b, escape sequence 960
\f, form-feed escape sequence 961
\n, newline escape sequence 27, 28, 961
\r, carriage-return escape sequence 28, 961
\t, horizontal tab escape sequence 28, 961
&, bitwise AND 973
&, boolean logical AND 110, 112
&&, conditional AND 110, 111
 truth table 110
&=, bitwise AND assignment operator 983
character 930
flag 956, 957
% conversion character 952`

`% SQL wildcard character` 740
`% remainder` 33, 34
`% format specifier` 954
`%=, remainder assignment operator` 81
`%A format specifier` 948
`%a format specifier` 948
`%B format specifier` 952
`%b format specifier` 113, 952, 953
`%C format specifier` 949
`%c format specifier` 949, 949
`%d format specifier` 32, 946, 946, 947
`%E format specifier` 947, 948
`%e format specifier` 947, 948
`%f format specifier` 57, 947, 948
`%G format specifier` 948
`%g format specifier` 948
`%H format specifier` 952
`%h format specifier` 953
`%n format specifier` 953
`%o format specifier` 946, 947
`%S format specifier` 949
`%s format specifier` 29, 946, 949
`%T format specifier` 950
`%t format specifier` 950
`%X format specifier` 946
`%x format specifier` 946
`- flag` 956
`+ flag` 956
`-(minus sign) formatting flag` 95
`+, addition` 33, 34
`++, preincrement/postincrement` 81
`+=, addition assignment operator` 81
`+=, string concatenation assignment operator` 452
`<, less than` 34
`<<, left shift` 973, 980
`<<=, left shift assignment operator` 983
`<=, less than or equal` 34
`>>, (diamond notation in generics)` 514
`>> diamond notation for generic type inference (Java SE 7)` 514
`=, assignment operator` 32
`-=, subtraction assignment operator` 81
`== to determine whether two references refer to the same object` 261
`==, is equal to` 34
`-> (arrow token in a lambda)` 551
`>, greater than` 34
`>=, greater than or equal to` 34
`>>, signed right shift` 973, 980
`>>=, signed right shift assignment operator` 983
`>>>, unsigned right shift` 973, 980
`>>>=, unsigned right shift assignment operator` 983
`|, bitwise inclusive OR` 973
`|, boolean logical inclusive OR` 110, 112

`|=, bitwise inclusive OR assignment operator` 983
`||, conditional OR` 110, 111
 truth table 111
`~ (bitwise complement)` 973
`~, bitwise complement` 974

Numerics

`0 flag` 154
`0 format flag` 196
`0x (hexadecimal prefix)` 957

A

abbreviating assignment expressions 81
`abs` method of `Math` 119
`absolute` method of `ResultSet` 760
absolute path 477, 479
absolute value 119
abstract class 266, 270, 271, 272, 291, 998
Abstract Factory design pattern 994, 995, 1006
abstract implementation 545
`abstract` keyword 271
abstract method 271, 273, 275, 352, 867, 898
abstract superclass 270, 867
Abstract Window Toolkit (AWT) 338, 788
Abstract Window Toolkit Event package 127
`AbstractButton` class 354, 357, 618, 623
 `addActionListener` method 357
 `addItemListener` method 360
 `isSelected` method 625
 `setMnemonic` method 623
 `setRolloverIcon` method 357
 `setSelected` method 624
`AbstractCollection` class 545
`AbstractList` class 545
`AbstractMap` class 545
`AbstractQueue` class 545
`AbstractSequentialList` class 545
`AbstractSet` class 545
`AbstractTableModel` class 755, 761
 `fireTableStructureChanged` method 761
`accept` method of functional interface `Consumer` (Java SE 8) 569
`accept` method of interface `BiConsumer` (Java SE 8) 573
`accept` method of interface `IntConsumer` (Java SE 8) 556

- access modifier 40, 41, 857
private 41, 199, 238
protected 199, 238
public 40, 199, 238
 access modifier in the UML
 - (private) 47
 access shared data 680
 accessibility 339
 accessor method 209
Account class (ATM case study) 823,
 826, 829, 831, 832, 839, 846, 847,
 848, 849, 850, 851, 878
 acquire a lock 664
 acquire the lock 664
 action 63, 70
 action expression in the UML 61, 836
 action key 384
 action of an object 836
 action state in the UML 61, 836
 action state symbol 61
ActionEvent class 348, 349, 353, 400
getActionCommand method 349,
 357
ActionEvent class (JavaFX) 809, 811
ActionListener interface 348, 353
actionPerformed method 348,
 352, 393, 400
actionPerformed method of interface
ActionListener 348, 352, 393,
 400
 activation in a UML sequence diagram
 850
 activity diagram 60, 61, 63, 92
do...while statement 98
for statement 92
if statement 63
if...else statement 64
 in the UML 70, 823, 836, 837, 854
 sequence statement 61
switch statement 103
while statement 70
 activity in the UML 60, 823, 835, 838
 actor in use case in the UML 821
 actual type arguments 589
 acyclic gradient 431
 adapter class 377
 Adapter Classes used to implement event
 handlers 381
 Adapter design pattern 994, 996, 999
add method
ArrayList<T> 191
ButtonGroup 363
JFrame 342
JMenuBar 623
JMenuBar 624
LinkedList<T> 518
List<T> 513, 516
add method of class **BigDecimal** 231
add rows or columns to a GridPane 803
addActionListener method
 of class **AbstractButton** 357
 of class **JTextField** 348
addAll method
Collections 519, 529
List 516
addFirst method of **LinkedList** 519
addItemListener method of class
AbstractButton 360
addition 33
 addition compound assignment operator,
 += 81
addKeyListener method of class
Component 384
addLast method of **LinkedList** 518
addListener method of interface
ObservableValue 812
addListSelectionListener method
 of class **JList** 369
addMouseListener method of class
Component 377
addMouseMotionListener method of
 class **Component** 377
addPoint method of class **Polygon** 426,
 428
addSeparator method of class **JMenu**
 624
addTab method of class **JTabbedPane**
 638
addTableModelListener method of
TableModel 755
addWindowListener method of class
Window 617
 aggregation in the UML 828
 Agile Alliance
 (www.agilealliance.org) 16
 Agile Manifesto
 (www.agilemanifesto.org) 16
 agile software development 16
 algorithm 1004
 in Java Collections Framework 519
 aligning decimal points in output 945
 alignment in a **VBox** (JavaFX) 796
Alignment property of a **VBox** (JavaFX)
 796
allClasses-frame.html generated by
javadoc 934
 alpha software 18
 alphabetizing 440
 analysis stage of the software life cycle 821
anchor field of class
GridBagConstraints 643
AND (in SQL) 746, 747
and method of class **BitSet** 984
and method of interface **Predicate**
 (Java SE 8) 563
Android 7
 Google Play 8
 operating system 7
 smartphone 7
*Android for Programmers: An App-Driven
 Approach* 8
 angle brackets (< and >) 588
 annotation
@Override 243
 anonymous inner class 194, 302, 348,
 366, 382, 812
 anonymous method (Java SE 8) 551
 anonymous methods 302
 Apache Software Foundation 7
 API (application programming interface)
 30, 118
 API documentation 128
 API links
 Deprecated 916
 Help 916
 Index 916
 Tree 916
append method of class
StringBuilder 455
 Apple Computer, Inc. 938
 application 21
 command-line arguments 121
Application class (JavaFX) 799
 launch method 799, 807
 start method 799, 807
 application programming interface (API)
 8, 118
apply method of functional interface
Function (Java SE 8) 565
applyAsDouble method of interface
ToDoubleFunction (Java SE 8) 575
applyAsInt method of interface
IntBinaryOperator (Java SE 8)
 558
applyAsInt method of interface
IntUnaryOperator (Java SE 8) 560
arc 422
arc angle 422
**arc width and arc height for rounded
 rectangles** 421
Arc2D class 403
 CHORD constant 432
 OPEN constant 432
 PIE constant 432
Arc2D.Double class 428
 architectural patterns 994, 1008, 1010
 archive files 227
args parameter 184
 argument index 946, 952, 960
 argument list 947
 argument promotion 125
 argument to a method 45
 arithmetic compound assignment
 operators 81
 arithmetic operators 33
 arithmetic overflow 73, 314
ArithmException class 232, 307,
 313
array 145, 475, 1011
 bounds checking 155
 ignoring element zero 157
length instance variable 147
 pass an array element to a method
 165
 pass an array to a method 165
 array-access expression 146
 array-creation expression 147
array initializer 149
 for multidimensional array 173
 nested 173
 array of one-dimensional arrays 173
ArrayBlockingQueue class 680, 691,
 706
size method 682
arraycopy method of class **System**
 186, 187
ArrayIndexOutOfBoundsException
 class 155, 158, 428
ArrayList<T> generic class 188, 511,
 527
add method 191
clear method 189
contains method 189, 191
get method 191
indexOf method 189
isEmpty method 210

A

- `ArrayList<T>` generic class (cont.)
 - `remove` method 189, 191
 - `size` method 191
 - `toString` method 607
 - `trimToSize` method 189
- `Arrays` class 186
 - `asList` method 517, 518
 - `binarySearch` method 186
 - `equals` method 186
 - `fill` method 186, 716
 - `parallelPrefix` method 723
 - `parallelSetAll` method 723
 - `parallelSort` method 721
 - `parallelSort` method (Java SE 8) 563
 - `sort` method 186, 563, 721
 - `stream` method (Java SE 8) 561, 562
 - `toString` method 471
- `Arrays` method `parallelSort` 188
 - arrow 61
 - arrow key 384
 - arrow token (`->`) in a lambda 551
 - arrowhead in a UML sequence diagram 850
 - artifact in the UML 990
 - ascending order 186
 - ASC in SQL 742, 743
 - ascend 416
 - ASCII (American Standard Code for Information Interchange) character set 104, 938
 - ASCII character set Appendix 897
 - `asList` method of `Arrays` 517, 518
 - `assert` statement 328, 898
 - assertion 328
 - `AssertionError` class 328
 - Assigning superclass and subclass
 - references to superclass and subclass variables 269
 - assignment operator, = 32, 36
 - assignment operators 81
 - associate
 - right to left 78
 - association (in the UML) 826, 827, 828, 859, 860
 - name 827
 - associativity of operators 33, 37, 84
 - left to right 37
 - right to left 33
 - asterisk (*) SQL wildcard character 739
 - asynchronous call 849
 - asynchronous event 314
 - ATM (automated teller machine) case study 816, 821
 - ATM class (ATM case study) 826, 827, 828, 831, 833, 835, 839, 846, 847, 848, 849, 858
 - ATM system 821, 822, 824, 825, 835, 839, 857
 - `ATMCaseStudy` class (ATM case study) 892
 - atomic operation 670, 786
 - attribute 857, 859, 860
 - compartment in a class diagram 833
 - declaration in the UML 833, 835
 - in the UML 5, 47, 826, 830, 832, 833, 835, 838, 865, 866
 - name in the UML 833
- attribute (cont.)
 - of a class 4
 - of an object 5
 - type in the UML 833
- @author javadoc tag 929
- A**uthor: note 929
 - author option 933
- `AuthorISBN` table of Books database 737
- `authorISBN` table of books database 735, 737
- `authors` table of books database 735
- auto commit state 786
- auto-unboxing 510
- autobox an int 592
- autoboxing 462, 510, 592
- `AutoCloseable` interface 216, 330, 752
 - `close` method 330
- autoincremented 735, 745
- automated teller machine (ATM) 816, 821
 - user interface 817
- automatic driver discovery (JDBC 4) 751
- automatic garbage collection 317
- automatic scrolling 369
- average 71, 73
- average method of interface
 - `DoubleStream` (Java SE 8) 575
- average method of interface
 - `IntStream` (Java SE 8) 557
- `await` method of interface `Condition` 699, 703
- `awaitTermination` method of interface `ExecutorService` 669
- `AWT` (Abstract Window Toolkit) 338, 788
 - components 339
- `AWTEvent` class 350

- B**
- B conversion character 952
- b conversion character 952
- background color 410, 412
- backing array 517
- backslash (\) 27, 960, 961
- `BalanceInquiry` class (ATM case study) 826, 829, 831, 832, 834, 836, 839, 847, 848, 849, 850, 858, 862, 863, 864
- Balking design pattern 994, 1005
- `BankDatabase` class (ATM case study) 826, 829, 831, 839, 841, 846, 847, 848, 849, 850, 851, 858, 860
- bar chart 152, 153
- bar of asterisks 152, 153
- base 973
- base class 235
- base of a number 461
- baseline of the font 414
- `BasePlusCommissionEmployee` class
- extends `CommissionEmployee` 282
- `BasicLatin` block 942
- `BasicStroke` class 403, 431, 432
- `CAP_ROUND` constant 432
- `JOIN_ROUND` constant 433
- batch file 483
- behavior 839
- of a class 4
- of a system 835, 836, 838, 848
- behavioral design patterns 993, 997, 1001, 1010
- beta software 18
- between method of class `Duration` 722
- `BiConsumer` functional interface (Java SE 8) 573, 580
- `accept` method 573
- bidirectional iterator 517
- bidirectional navigability in the UML 858
- `BigDecimal` class 78, 96, 230, 808
- `add` method 231
- `ArithmaticException` class 232
- `multiply` method 231
- ONE constant 231
- `pow` method 231
- `setScale` method 232
- TEN constant 231
- `valueOf` method 230
- ZERO constant 231
- `BigInteger` class 708
- binary
- base 2 number system 964
- binary file 476
- binary operator 32, 33, 112
- binary search algorithm 527
- `BinaryOperator` functional interface (Java SE 8) 551
- `binarySearch` method
- of `Arrays` 186, 188
- of `Collections` 519, 527, 529
- bit manipulation 973
- `BitSet` class 973, 983
- `and` method 984
- `clear` method 984
- `equals` method 984
- `get` method 984
- `or` method 984
- `set` method 983
- `size` method 984
- `toString` method 984
- `xor` method 984
- bitwise AND (&) 973
- Bitwise AND, bitwise inclusive OR, bitwise exclusive OR (\\) and bitwise complement operators 976
- bitwise assignment operators 983
- `^` (bitwise exclusive OR) 983
- `&` (bitwise AND) 983
- `<<=` (left shift) 983
- `>>=` (signed right shift) 983
- `>>>=` (unsigned right shift) 983
- `|` (bitwise inclusive OR) 983
- bitwise complement (~) operator 973, 974, 980
- bitwise exclusive OR (^) operator 973, 980
- bitwise inclusive OR (|) operator 973
- bitwise operators 110, 973, 974
- `^` (bitwise exclusive OR) 973
- `&` (bitwise AND) 973
- `<<` (left shift) 973
- `>>` (signed right shift) 973
- `>>>` (unsigned right shift) 973
- `|` (bitwise inclusive OR) 973
- `~` (complement) 973
- bitwise shift operations 981
- blank line 22
- block 65, 77

- block increment of a `JSlider` 613
`blocked` state 657, 665
BlockingQueue interface 680
 `put` method 680, 681
 `take` method 680, 682
body
 of a class declaration 23
 of a loop 70
 of a method 23
 of an `if` statement 34
Bohm, C. 60
BOLD constant of class `Font` 414, 414
book-title capitalization 337, 354
books database 735
 table relationships 738
Boolean
 attribute in the UML 831
 class 510
boolean 105
 expression 66, 906
 promotions 126
boolean logical AND, & 110, 112
boolean logical exclusive OR, ^ 110, 112
 truth table 112
boolean logical inclusive OR, | 112
boolean primitive type 66, 898, 899,
 906
border of a `JFrame` 616
BorderLayout class 377, 387, 388, 391,
 400
 CENTER constant 377, 391, 393
 EAST constant 377, 391
 NORTH constant 377, 391
 SOUTH constant 377, 391
 WEST constant 377, 391
BOTH constant of class
 `GridBagConstraints` 644
bounded buffer 691
bounding rectangle 420, 422, 613
bounds checking 155
Box class 399, 639, 641
 `createGlue` method 642
 `createHorizontalBox` method
 400, 641
 `createHorizontalGlue` method
 642
 `createHorizontalStrut` method
 641
 `createRigidArea` method 642
 `createVerticalBox` method 641
 `createVerticalGlue` method 642
 `createVerticalStrut` method
 641
 X_AXIS constant 642
 Y_AXIS constant 642
boxed method of interface `InputStream`
 (Java SE 8) 580
boxing 189
boxing conversion 510, 592
BoxLayout class 400, 639
BoxLayout layout manager 639
braces ({ and }) 65, 77, 90, 98, 149
 not required 102
braille screen reader 339
branch 1000
break 898
break mode 903
break statement 102, 108
breakpoint 901
 inserting 903, 905
 listing 913
 removing 913
Bridge design pattern 994, 996, 1000
brightness 412
brittle software 255
buffer 504, 673
buffered I/O 504
BufferedImage class 432
 `createGraphics` method 432
 TYPE_INT_RGB constant 432
BufferedInputStream class 505
BufferedOutputStream class 504,
 1007
 `flush` method 504
BufferedReader class 505
BufferedWriter class 505
Builder design pattern 995
building-block approach to creating
 programs 5
bulk operation 510
business publications 18
button 334, 354
Button class (JavaFX) 805
button label 354
ButtonGroup class 360, 618, 625
 `add` method 363
byte-based stream 476
Byte class 510
byte keyword 899
byte primitive type 98, 898, 973
 promotions 126
ByteArrayInputStream class 505
ByteArrayOutputStream class 505
bytecode 10, 25
bytecode verifier 11
- ## C
- CachedRowSet** interface 767
 `close` method 769
calculations 37, 60
Calendar class 951
 `getInstance` method 952
call-by-reference 166
call-by-value 166
call method of interface `Callable` 726
Callable interface 726
 `call` method 726
CallableStatement interface 785
camel case 31
camera 8
cancel method of class `SwingWorker`
 721
CANCEL_OPTION constant of
 `JFileChooser` 501
CAP_ROUND constant of class
 `BasicStroke` 432
capacity method
 of class `StringBuilder` 452
capacity of a `StringBuilder` 451
capturing lambdas 556
card games 158
Card Shuffling and Dealing
 with `Collections` method
 `shuffle` 523
caretaker object 998
carriage return 28
carry bit 971
Cascading 789
Cascading Style Sheets (CSS) 789, 789
case keyword 102, 898
case sensitive 23
 Java commands 13
CashDispenser class (ATM case study)
 826, 827, 828, 831, 832, 839, 851,
 876
casino 133
cast
 downcast 268
 operator 77, 126
catch
 a superclass exception 317
 an exception 309
catch
 block 311, 313, 314, 318, 321, 322
 clause 311, 898
 keyword 311
Catch block 157
catch handler
 multi-catch 312
catch-or-declare requirement 316
cd to change directories 24
ceil method of `Math` 120
cell in a `GridPane` 799
CENTER constant
 `BorderLayout` 377, 391, 393
 `FlowLayout` 391
 `GridBagConstraints` 643
center mouse button click 380
centered 389
Chain-of-Responsibility design pattern
 994, 997, 1002
chained exception 325
change directories 24
change the default layout (JavaFX Scene
 Builder) 795, 803
changed method of interface
 `ChangeListener` (JavaFX) 808
ChangeEvent class 616
ChangeListener interface 616
 `stateChanged` method 616
ChangeListener interface (JavaFX)
 801, 808, 812
changing look-and-feel of a Swing-based
 GUI 632
char
 array 439
 keyword 898, 899
 primitive type 31, 98
 promotions 126
character
 constant 104
 literal 437
character-based stream 476
Character class 437, 460, 510
 `charValue` method 462
 `digit` method 461
 `forDigit` method 461
 `isDefined` method 460
 `isDigit` method 460
 `isJavaIdentifierPart` method
 460
 `isJavaIdentifierStart` method
 460
 `isLetter` method 460
 `isLetterOrDigit` method 460

Character class (cont.)
 isLowerCase method 460
 isUpperCase method 460
 static conversion methods 461
 toLowerCase method 461
 toUpperCase method 460
character encoding 937
character set 938
character string 24
CharArrayReader class 506
CharArrayWriter class 506
charAt method
 of class *String* 439
 of class *StringBuilder* 454
CharSequence interface 471
charValue method of class *Character* 462
checkbox 354, 360
checkbox label 360
checked exception 315
Checking with assert that a value is within range 329
child window 612, 633, 635, 636
CHORD constant of class *Arc2D* 432
circular buffer 692
CJK Unified Ideographs block 942
class 4, 833, 839, 843, 857
 anonymous inner class 194
 class keyword 40
 constructor 44, 50, 859
 data hiding 48
 declaration 22
 default constructor 52
 file 25
 get method 202
 instance variable 5
 name 22, 859
 nested class 194
 set method 202
class-average problem 71, 75
class cannot extend a final class 289
Class class 262, 287, 343, 760
 getName method 262, 287
 getResource method 343
class diagram
 for the ATM system model 829, 853
 in the UML 823, 826, 828, 832,
 839, 857, 860, 864, 865, 866
.class file 10, 25
 separate one for every class 200
class hierarchy 235, 271
class instance creation expression 44, 52,
 196
 diamond notation (<>) 191, 192
class keyword 22, 40, 898
class library 236
class loader 10, 227, 343
class method 119
class name
 fully qualified 48
class names
 camel case naming 40
class variable 120, 216
classwide information 216
ClassCastException class 509
Classes
 AbstractButton 354, 357, 618,
 623
 AbstractCollection 545

Classes (cont.)
 AbstractList 545
 AbstractMap 545
 AbstractQueue 545
 AbstractSequentialList 545
 AbstractSet 545
 AbstractTableModel 755, 761
 ActionEvent 348, 349, 353, 400
 ActionEvent (JavaFX) 809, 811
 Application (JavaFX) 799
 Arc2D 403
 Arc2D.Double 428
 ArithmetricException 232, 307
 ArrayBlockingQueue 680, 691,
 706
 ArrayIndexOutOfBoundsException
 155, 158
 ArrayList<T> 188, 189, 191, 511,
 512, 527
 Arrays 186
 AssertionError 328
 AWTEvent 350
 BasicStroke 403, 431, 432
 BigDecimal 78, 96, 230, 808
 BigInteger 708
 BitSet 973
 Boolean 510
 BorderLayout 377, 387, 388, 391,
 400
 Box 399, 639, 641
 BoxLayout 400, 639
 BufferedImage 432
 BufferedInputStream 505
 BufferedOutputStream 504
 BufferedReader 505
 BufferedWriter 505
 ButtonGroup 360, 618, 625
 Byte 510
 ByteArrayInputStream 505
 ByteArrayOutputStream 505
 Calendar 951
 ChangeEvent 616
 Character 437, 455, 460, 510
 CharArrayReader 506
 CharArrayWriter 506
 Class 262, 287, 343, 760
 Collections 511, 591
 Collector (Java SE 8) 563
 Color 403
 CompletableFuture class 726
 Component 339, 372, 405, 406,
 617, 648
 ComponentAdapter 378
 ComponentListener 388
 ConcurrentHashMap 706
 ConcurrentLinkedDeque 706
 ConcurrentSkipListMap 706
 ConcurrentSkipListSet 706
 Container 339, 370, 388, 396, 397
 ContainerAdapter 378
 CopyOnWriteArrayList 706
 CopyOnWriteArraySet 706
 DataInputStream 504
 DataOutputStream 504
 Date 951
 DelayQueue 706
 Double 510, 606
 DoubleProperty (JavaFX) 812
 DriverManager 752

Classes (cont.)
 Ellipse2D 403
 Ellipse2D.Double 428
 Ellipse2D.Float 428
 EmptyStackException 533
 EnumSet 215
 Error 314
 EventListenerList 352
 Exception 314
 ExecutionException 710
 Executors 660
 FileReader 506
 Files 477, 577
 FilterInputStream 504
 FilterOutputStream 504
 Float 510
 FlowLayout 342, 388
 FocusAdapter 378
 Font 359, 403, 414
 FontMetrics 403, 416
 Formatter 477, 945
 Frame 616
 FXMLLoader (JavaFX) 801, 807,
 810, 811
 GeneralPath 403, 433
 GradientPaint 403, 431
 Graphics 382, 403, 428
 Graphics2D 403, 428, 432
 GridBagConstraints 643, 648,
 649
 GridBagLayout 639, 642, 644,
 649
 GridLayout 388, 395
 GridPane (JavaFX) 799
 HashMap 537
 HashSet 534
 Hashtable 537
 IllegalMonitorStateException
 684, 699
 ImageIcon 343
 ImageView (JavaFX) 791
 IndexOutOfBoundsException 158
 InetAddress 1007
 InputEvent 373, 380, 384
 InputMismatchException 308
 InputStream 503
 InputStreamReader 506
 Instant (Java SE 8) 729
 Integer 337, 510, 606
 InterruptedException 661
 ItemEvent 360, 363
 JApplet 617
 JButton 338, 354, 357, 393
 JCheckBox 338, 357
 JCheckBoxMenuItem 617, 618,
 624
 JColorChooser 410
 JComboBox 338, 363, 644
 JComponent 339, 340, 342, 352,
 364, 367, 381, 397, 403, 405
 JDesktopPane 633
 JDialog 623
 JFileChooser 500
 JFrame 616
 JInternalFrame 633, 635
 JLabel 338, 340
 JList 338, 367
 JMenu 617, 624, 635
 JMenuBar 617, 624, 635

- Classes (cont.)
- JMenuItem 618, 635
 - JOptionPane 335
 - JPanel 338, 381, 388, 397, 613
 - JPasswordField 344, 349
 - JPopupMenu 625
 - JProgressBar 717
 - JRadioButton 357, 360, 363
 - JRadioButtonMenuItem 617, 618, 625
 - JScrollPane 369, 372, 400, 401
 - JSlider 612, 613, 616
 - JTabbedPane 636, 642
 - JTable 754
 - JTextArea 387, 398, 400, 644, 647
 - JTextComponent 344, 347, 398, 400
 - JTextField 338, 344, 348, 352, 398
 - JToggleButton 357
 - KeyAdapter 378
 - KeyEvent 353, 384
 - Label (JavaFX) 791
 - Line2D 403, 432
 - Line2D.Double 428
 - LinearGradientPaint 431
 - LineNumberReader 506
 - LinkedBlockingDeque 706
 - LinkedBlockingQueue 706
 - LinkedList 511
 - LinkedTransferQueue 706
 - ListSelectionEvent 367
 - ListSelectionModel 369
 - Long 510
 - Matcher 437, 471
 - Math 119
 - MouseAdapter 377, 378
 - MouseEvent 353, 373, 628
 - MouseMotionAdapter 378, 381
 - MouseWheelEvent 374
 - Node (JavaFX) 790
 - Number 606
 - NumberFormat 230, 722, 800, 808
 - Object 216
 - ObjectInputStream 492
 - ObjectOutputStream 492
 - Optional class (Java SE 8) 570
 - OptionalDouble 557, 575
 - OutputStream 503
 - OutputStreamWriter 506
 - Parent (JavaFX) 802, 807
 - Paths 477
 - Pattern 437, 471
 - PipedInputStream 503
 - PipedOutputStream 503
 - PipedReader 506
 - PipedWriter 506
 - Point 382
 - Polygon 403, 425
 - PrintStream 504
 - PrintWriter 506
 - PriorityBlockingQueue 706
 - PriorityQueue 533
 - Properties 541
 - RadialGradientPaint 431
 - Reader 505
 - Rectangle2D 403
 - Rectangle2D.Double 428
 - ReentrantLock 698, 700
 - RoundRectangle2D 403
- Classes (cont.)
- RoundRectangle2D.Double 428, 432
 - RowFilter 766
 - RowSetFactory 767
 - RowSetProvider 767
 - RuntimeException 315
 - Scanner 31
 - Scene (JavaFX) 790, 799, 807, 808
 - SecureRandom 128
 - Short 510
 - Slider (JavaFX) 798, 800
 - Socket 1006
 - SQLException 752
 - SQLFeatureNotSupportedException 760
 - Stack 531
 - StackTraceElement 324
 - Stage (JavaFX) 790, 799, 807, 808
 - String 437
 - StringBuffer 452
 - StringBuilder 437, 451
 - StringIndexOutOfBoundsException 447
 - StringReader 506
 - StringWriter 506
 - SwingUtilities 632
 - SwingWorker 707
 - SynchronousQueue 706
 - SystemColor 431
 - TableModelEvent 766
 - TableRowSorter 766
 - TextField (JavaFX) 800
 - TexturePaint 403, 431, 432
 - Throwable 314, 324
 - TreeMap 537
 - TreeSet 534
 - Types 754
 - UIManager 629
 - UnsupportedOperationException 517
 - Vbox (JavaFX) 795
 - Vector 511
 - Window 616, 617
 - WindowAdapter 378, 766
 - Writer 505
 - class-instance creation expression 813
 - ClassName, this 623
 - CLASSPATH
 - environment variable 25, 227 - classpath 227, 751
 - classpath command-line argument
 - to java 228
 - to javac 227 - CLASSPATH problem 13
 - clear debugger command 913
 - clear method
 - of ArrayList<T> 189
 - of BitSet 984
 - of List<T> 517
 - of PriorityQueue 533 - clearRect method of class Graphics 419
 - click a button 344, 801
 - click count 378
 - click the scroll arrows 366
 - client
 - object 1007
 - of a class 48, 839, 848
- client code 267
- client tier 1009
- clone method of Object 262
- cloning objects
- deep copy 262
 - shallow copy 262
- close a window 340, 344, 801
- close method
- of CachedRowSet 769
 - of Connection 754
 - of Formatter 484
 - of interface Connection 754
 - of interface ResultSet 754
 - of interface Statement 754
 - of JdbcRowSet 769
 - of ObjectOutputStream 497
 - of ResultSet 754
 - of Statement 754
- close method of interface
- AutoCloseable 330
- closed polygons 425
- closePath method of class
- GeneralPath 435
- cloud computing 17
- code 6
- code value 939, 942
- coin tossing 129
- collaboration diagram in the UML 823
- collaboration in the UML 845, 846, 847, 849
- collect method of interface Stream
- (Java SE 8) 563, 573, 574, 580
- collection 188, 508
- collection implementation 544
- Collection interface 509, 510, 514, 519
- contains method 514
 - iterator method 514
- collections
- synchronized collection 511
 - unmodifiable collection 511
- Collections class 511, 591
- addAll method 519, 529
 - binarySearch method 519, 527, 529
 - copy method 519, 526
 - disjoint method 519, 529
 - fill method 519, 525
 - frequency method 519, 529
 - max method 519, 526
 - min method 519, 526
 - reverse method 519, 525
 - reverseOrder method 521
 - shuffle method 519, 523, 525
 - sort method 520
 - wrapper methods 511
- collections framework 508
- Collections methods reverse, fill, copy, max and min 526
- Collector interface (Java SE 8) 563
- Collectors class (Java SE 8) 563
- groupingBy method 573, 574, 578, 580
 - toList method 563
- collision in a hashtable 538
- color 403
- Color class 403
- getBlue method 407, 409
 - getColor method 407

Color class (cont.)

 getGreen method 407, 409

 getRed method 407, 409

 setColor method 407

Color constant 406, 409

 color manipulation 405

 color swatches 412

 color-chooser dialog 412

 column 173, 734, 735

 column number in a result set 740

 columns of a two-dimensional array 173

 combo box 334, 363

 comma (,) 94

 comma (,) formatting flag 96

 comma-separated list 93

 of parameters 122

 command button 354

 Command design pattern 994, 997, 1002

 command line 24

 command-line argument 121, 184

Command Prompt 10, 24

 command window 12, 24

 comment

 end-of-line (single-line), // 22

 Javadoc 22

CommissionEmployee class derived
 from **Employee** 280

 commit a transaction 786

commit method of interface
 Connection 786

 Common Programming Errors overview
 xxxi

 communication diagram in the UML
 823, 848, 849

Comparable<T> interface 300, 443,
 520, 590

 compareTo method 520, 591

Comparator interface 520, 521, 566,
 566

 compare method 522

 thenComparing method (Java SE
 8) 571

Comparator object 520, 526, 535, 537

 in **sort** 520

 compare method of interface
 Comparator 522

compareTo method
 of class **String** 441, 443
 of **Comparable** 520

compareTo method of **Comparable**<T>
 591

 comparing **String** objects 440

 comparison operator 300

 compartment in a UML class diagram 46

 compilation errors 25

 compile 24

 compile a program 10

compile method of class **Pattern** 471

 compile-time type safety 513

 compiler options
 -d 224

 compile-time type safety 585

 compiling an application with multiple
 classes 46

CompletableFuture class (Java SE 8)
 726

 runAsync method 730

 supplyAsync method 729

 complex curve 433

 component 4, 372

Component class 339, 372, 405, 406,
 412, 617, 648, 999, 1001

 addKeyListener method 384

 addMouseListener method 377

 addMouseMotionListener
 method 377

 getMinimumSize method 616

 getPreferredSize method 615

 repaint method 383

 setBackground method 412

 setBounds method 388

 setFont method 359

 setLocation method 388, 617

 setSize method 388, 617

 setVisible method 393, 617

 component diagram in the UML 990

 component in the UML 990

 component of an array 146

ComponentAdapter class 378

ComponentListener interface 378, 388

 composing lambda expressions 560

 Composite design pattern 994, 996, 1001

 composite primary key 737, 738

 composite structure diagram in the UML
 991

 composition 210, 236, 238, 827, 828,
 853

 in the UML 827

 compound assignment operators 81, 83

 compound interest 94

 computerized scientific notation 947

concat method of class **String** 448

 concatenate strings 219

 concatenation 123

 concordance 576

 concrete class 271

 concrete subclass 276

CONCUR_READ_ONLY constant 759

CONCUR_UPDATABLE constant 759

 Concurrency API 655

 concurrency APIs 655

 concurrency design patterns 994, 1005

 concurrency problem 1005

 concurrent access to a **Collection** by
 multiple threads 544

 concurrent operations 654

 concurrent programming 655

 concurrent threads 680

ConcurrentHashMap class 706

ConcurrentLinkedDeque class 706

ConcurrentSkipListMap class 706

ConcurrentSkipListSet class 706

 condition 34, 98

Condition interface 699, 700

 await method 699, 703

 signal method 699

 signalAll method 699

 condition object 699

 conditional AND, && 110, 111, 112

 truth table 110

 conditional expression 66

 conditional operator, ?: 66

 conditional OR, || 110, 111

 truth table 111

 confusing the equality operator == with
 the assignment operator = 36

 connect to a database 750

 connected lines 425

- conversion characters (cont.)
 a 948
 B 952
 b 952, 953
 C 949
 c 949
 d 946
 E 947, 948
 e 947, 948
 f 947, 948
 G 948
 g 948
 H 952
 h 953
 n 953
 o 946
 S 949
 s 949
 T 950
 t 950
 X 946
 x 946
- conversion suffix characters 950
 A 950
 a 950
 B 950
 b 950
 c 950
 D 950
 d 950
 e 951
 F 950
 H 951
 I 951
 j 951
 k 951
 l 951
 M 951
 m 950
 P 951
 p 951
 R 950
 r 950
 S 951
 T 950
 Y 951
 y 951
 Z 951
- convert
 a binary number to decimal 969
 a hexadecimal number to decimal 969
 an octal number to decimal 969
 between number systems 461
- coordinate system 403, 405
- coordinates (0, 0) 403
- copy method of *Collections* 519, 526
- copying files 477
- copying objects
 deep copy 262
 shallow copy 262
- CopyOnWriteArrayList* class 706
- CopyOnWriteArraySet* class 706
- core package 25
- cos* method of *Math* 120
- cosine 120
- count method of interface *IntStream*
 (Java SE 8) 557
- counter 71
- counter-controlled repetition 71, 71, 76,
 87, 88, 89
- coupling 1003, 1003
- cp command line argument
 to *java* 228
- craps (casino game) 133
- create a package 222
- create a reusable class 222
- create an object of a class 44
- create and use your own packages 222
- createGlue* method of class *Box* 642
- createGraphics* method of class
BufferedImage 432
- createHorizontalBox* method of class
Box 400, 641
- createHorizontalGlue* method of
 class *Box* 642
- createHorizontalStrut* method of
 class *Box* 641
- createJdbcRowSet* method of interface
RowSetFactory 767
- createRigidArea* method of class *Box*
 642
- createStatement* method of
Connection 753, 759
- createVerticalBox* method of class
Box 641
- createVerticalGlue* method of class
Box 642
- createVerticalStrut* method of class
Box 641
- creating and initializing an array 148
- creating files 477
- creational design patterns 993, 994, 999,
 1006, 1010
- CSS (Cascading Style Sheets) 789
- <*Ctrl*-d> 101
- Ctrl*/key 370, 387
- ctrl key 101
- <*Ctrl*-z> 101
- currentThread* method of class
Thread 661, 666
- cursor 24, 26
- curve 433
- custom drawing area 381
- customized subclass of class *JPanel* 381
- cyclic gradient 431
- D**
- d compiler option 224
- d option 933
- dangling-else problem 65
- dashed lines 428
- data hiding 48
- data integrity 209
- data source 549
- data structure 145, 1011
- database 733, 739
 table 734
- database management system (DBMS)
 733
- DataInput* interface 504
- DataInputStream* class 504
- DataOutput* interface 504
 writeBoolean method 504
 writeByte method 504
 writeBytes method 504
 writeChar method 504
- DataOutput* interface (cont.)
 writeChars method 504
 writeDouble method 504
 writeFloat method 504
 writeInt method 504
 writeLong method 504
 writeShort method 504
 writeUTF method 504
- DataOutputStream* class 504
- date and time compositions 950
- Date* class 951
- date formatting 946
- Date/Time API 188, 199
- Date/Time API package 128
- DBCS (double byte character set) 940
- DB2 733
- dead state 657
- deadlock 659, 703, 1005
- dealing 158
- debugger 901
 break mode 903
 breakpoints 901
 clear command 913
 cont command 904
 defined 901
 exit command 909
 -g compiler option 902
 inserting breakpoints 903
 jdb command 902
 logic error 901
 next command 909
 print command 905, 906
 run command 903, 905
 set command 905, 906
 step command 907
 step up command 908
 stop command 903, 905
 suspending program execution 905
 unwatch command 910, 911
 watch command 909
- decimal (base 10) number system 964
- decimal integer 946
- decimal integer formatting 32
- decision 34, 62
 symbol in the UML 62, 838
- declaration
 class 22
 import 30, 31
 method 23
- declarative programming 790
- Decorator design pattern 994, 996, 1006,
 1007
- decrement a control variable 91
- decrement operator, -- 82
- dedicated drawing area 381
- deep copy 262
- default case in a switch 102, 104, 132
- default constructor 52, 208, 242
- default exception handler 324
- default initial value 45
- default interface methods (Java SE 8)
 301
- default keyword 898
- default layout of the content pane 400
- default locale 809
- default method in an interface (Java SE
 8) 550, 581
- default method of an interface (Java SE
 8) 548

default methods in interfaces (Java SE 8) 301
default package 48, 224
default upper bound (*Object*) of a type parameter 596
default value 45, 85
define a custom drawing area 381
degree 422
 Deitel Resource Centers 18
DelayQueue class 706
 delegation event model 351
delete method of class *StringBuilder* 457
DELETE SQL statement 739, 747
deleteCharAt method of class *StringBuilder* 457
 deleting directories 477
 deleting files 477
 delimiter for tokens 463
 delimited string 463
 dependent condition 111
 deployment diagram in the UML 990
Deposit class (ATM case study) 826, 829, 831, 832, 839, 847, 848, 855, 858, 862, 863
DepositSlot class (ATM case study) 826, 827, 828, 831, 839, 848, 859
 deprecated APIs 30
~~@deprecated~~ javadoc tag 932
 Deprecated link in API 916
~~deprecated-list.html~~ generated by javadoc 935
 Deprecated note 932
deprecation flag 30
 derived class 235
 descending order 186
 descending sort (DESC) 742
 descent 416
 descriptive words and phrases 831, 832
 deserialized object 492
 design pattern 17, 992, 993, 995, 996, 998, 999, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1010
Design Patterns, Elements of Reusable Object-Oriented Software 993
 design process 6, 816, 822, 840, 845
 design specification 822
 diacritic 939
 dialog 335
 dialog box 335, 623
 Dialog font 414
 DialogInput font 414
 diamond in the UML 61
 diamond notation 514
 diamond notation (<>) 191, 191, 192
 dice game 133
 digit 31, 462, 464, 964
digit method of class *Character* 461
 direct superclass 235, 237
 directories
 creating 477
 getting information about 477
 manipulating 477
DIRECTORIES_ONLY constant of *JFileChooser* 500
 directory 477
 separator 227
DirectoryStream interface 477
 disconnected RowSet 767
disjoint method of class *Collections* 519, 529
 disk 475
 disk I/O completion 314
 dismiss a dialog 336
 dispatch
 a thread 658
 an event 353
 display a line of text 24
 display monitor 403
 display output 37
dispose method of class *Window* 617
DISPOSE_ON_CLOSE constant of interface *WindowConstants* 617
 distance between values (random numbers) 133
distinct method of interface *Stream* (Java SE 8) 572
 divide by zero 12, 307
 division 33
 division compound assignment operator, /= 81
DO_NOTHING_ON_CLOSE constant of interface *WindowConstants* 616
do...while repetition statement 62, 97, 98, 898
 document 612, 633
 documentation comments 924
 dollar signs (\$) 23
 dot(.) separator 44, 95, 119, 217, 428
 dotted line in the UML 61
 double-byte character set (DBCS) 940
 (double) cast 77
Double class 510, 606
 double-precision floating-point number 53
 double primitive type 31, 53, 75, 898, 899
 promotions 126
 double quotes, " 24, 27, 28
 double-selection statement 62
DoubleProperty class (JavaFX) 812
doubles method of class *SecureRandom* (Java SE 8) 580
DoubleStream interface (Java SE 8) 554, 574
 average method 575
 reduce method 575
 sum method 575
doubleValue method of class *Number* 607
 downcast 287, 509
 downcasting 268
 downstream *Collector* (Java SE 8) 574
 drag the scroll box 366
 dragging the mouse to highlight 400
draw method of class *Graphics2D* 431
 draw shapes 403
draw3DRect method of class *Graphics* 419, 422
drawArc method of class *Graphics* 422
 drawing color 407
 drawing on the screen 405
drawLine method of class *Graphics* 419
drawOval method of class *Graphics* 419, 422
drawPolygon method of class *Graphics* 425, 427
drawPolyline method of class *Graphics* 425, 427
drawRect method of class *Graphics* 419, 432
drawRoundRect method of class *Graphics* 420
drawString method of class *Graphics* 409
driver class 43
DriverManager class 752
 getConnection method 752
 drop-down list 338, 363
 dummy value 74
Duration class
 between method 722
 toMillis method 722
 dynamic binding 270, 286
 dynamic content 8
 dynamic resizing 145

E

eager 552
 eager stream operation (Java SE 8) 557
 eager terminal operation 559
EAST constant
 of class *BorderLayout* 377, 391
 of class *GridBagConstraints* 643
 echo character of class *JPasswordField* 344
 Eclipse
 demonstration video (www.deitel.com/books/jhttp9) 21
 Eclipse (www.eclipse.org) 9
 Eclipse Foundation 7
 edit a program 9
 editor 9
 effectively final local variables (Java SE 8) 556
 efficient (Unicode design principle) 938
 element of an array 146
 element of chance 128
 elided UML diagram 826
 eligible for garbage collection 220
 eliminate resource leaks 318
Ellipse2D class 403
Ellipse2D.Double class 428
Ellipse2D.Float class 428
 ellipsis (...) in a method parameter list 182
else keyword 63, 898
emacs 9
 embedded system 7
 embedded version of Java DB 747
Employee abstract superclass 275
Employee class hierarchy test program 283
Employee class that implements *Payable* 295
 empty statement (a semicolon, ;) 37, 66, 98
 empty string 349, 439
EmptyStackException class 533
 encapsulation 5
 end cap 431
End key 384
 "end of data entry" 74

end-of-file (EOF)
 indicator 101
 key combinations 484
 marker 475
 end-of-line (single-line) comment, // 22
endsWith method of class `String` 444
 enhanced for statement 163
ensureCapacity method of class
`StringBuilder` 453
`Enter (or Return)` key 24, 352
 entity-relationship diagram 738
enum 136
 constant 213
 constructor 213
 declaration 213
`EnumSet` class 215
 keyword 136, 898
 values method 214
enum type 136
 enumeration constant 136
`EnumSet` class 215
 range method 215
 environment variable
 CLASSPATH 25
 PATH 25
`EOFException` class 500
 equal likelihood 131
 equality operator == to compare `String`
 objects 441
 equality operators 34
equals method
 of class `Arrays` 186
 of class `BitSet` 984
 of class `Object` 261
 of class `String` 441, 443
equalsIgnoreCase method of class
`String` 441, 443
 erasure 590, 593
Error class 314
 Error-Prevention Tips overview xxxi
 escape character 27, 745
 escape sequence 27, 31, 481, 960, 961
 \, backslash 28
 \" , double-quote 28
 \t, horizontal tab 28
 newline, \n 27, 28
 event 301, 344, 405, 791, 800, 835
 event classes 350
 event-dispatch thread (EDT) 405, 707
 event driven 344, 800
 event-driven process 405
 event culling 800
 event handler 301, 344, 791, 801
 implement with a lambda 581, 813
 lambda 581
 event handling 344, 347, 352, 791, 801
 event source 349
F
 event ID 353
 event listener 301, 351, 377
 adapter class 377
 interface 347, 348, 351, 352, 353,
 372, 377, 801
 event object 351
 event registration 348
 event source 349, 351
`EventHandler<ActionEvent>`
 interface (JavaFX) 811
`EventListenerList` class 352
EventObject class
`getSource` method 349
exception 157, 305
 handler 157
 handling 155
 parameter 158
Exception class 314
 exception handler 311
 exception parameter 311
 Exceptions 158
`IndexOutOfBoundsException` 158
execute 11
execute method
 of `JdbcRowSet` 769
execute method of the `Executor`
 interface 659, 663
executeQuery method
 of `PreparedStatement` 776
 of `Statement` 753
executeUpdate method of interface
`PreparedStatement` 776
 executing an application 13
 execution-time error 12
ExecutionException class 710
Executor interface 659
 execute method 659, 663
Executors class 660
 newCachedThreadPool method
 661
ExecutorService interface 660, 726
 awaitTermination method 669
 shutdown method 663
 submit method 726
exists method of class `Files` 478
exit debugger command 909
exit method of class `System` 318, 483
 exit point
 of a control statement 62
 exiting a `for` statement 108
exp method of `Math` 120
 expanded submenu 623
 explicit conversion 77
 exponential format 946
 exponential method 120
 exponential notation 947
 exponentiation operator 95
 expression 32
 extend a class 235
 extends keyword 239, 250, 898
 extensibility 267
 extensible language 39
 extension mechanism
 extending Java with additional class
 libraries 227
 external event 372
 external iteration 549, 579
F
 Facade design pattern 994, 996, 1007
 facade object 1007
 Facebook 7
 factory 1006
 factory method 999
 Factory Method design pattern 994, 995,
 999
 fail fast iterator 514
 fairness policy of a lock 698
false keyword 34, 66, 898
 fatal runtime error 12
 fault tolerant 305
 fault-tolerant program 157
 feature-complete 18
 field
 default initial value 45
 Field Detail section in API 921
 field of a class 120, 138
 Field Summary section in API 919
 field width 95, 946, 954
 file 475
File class
`toPath` method 501
 used to obtain file and directory
 information 479
FileInputStream class 544
FileNotFoundException class 483
FileOutputStream class 543, 1006
FileReader class 506
 files
 copying 477
 creating 477
 getting information about 477
 manipulating 477
 reading 477
Files class 477, 577
 exists method 478
`getLastModifiedTime` method
 478
 isDirectory method 478
 lines method (Java SE 8) 577
 newDirectoryStream method 478
 newOutputStream method 495,
 498
 size method 478
FILES_AND_DIRECTORIES constant of
`JFileChooser` 500
FILES_ONLY constant of
`JFileChooser` 500
FileWriter class 506
fill method
 of class `Arrays` 186, 187
 of class `Collections` 519, 525
 of class `Graphics2D` 431, 432, 435
fill method of class `Arrays` 716
 fill pattern 432
 fill texture 432
 fill with color 403
fill3DRect method of class `Graphics`
 419, 422
fillArc method of class `Graphics` 422
 filled-in shape 432
 filled rectangle 407
 filled three-dimensional rectangle 419
fillOval method of class `Graphics`
 383, 419, 422
fillPolygon method of class
`Graphics` 425, 428
fillRect method of class `Graphics`
 407, 419, 432
fillRoundRect method of class
`Graphics` 420
 filter a stream 504
 filter elements of a stream (Java SE 8) 559
filter method of interface `IntStream`
 (Java SE 8) 559
filter method of interface `Stream`
 (Java SE 8) 563, 566
FilterInputStream class 504

FilterOutputStream class 504
final
 class 289
 keyword 104, 120, 151, 221, 288, 664, 898
 local variable 366
 method 288
 variable 151
final release 18
final state in the UML 61, 836
final value 87
finalize method 216, 262
finally
 block 311, 317, 703
 clause 317, 898
 keyword 311
find method of class **Matcher** 471
findFirst method of interface **Stream** (Java SE 8) 570
fireTableStructureChanged
 method of **AbstractTableModel** 761
first method of **SortedSet** 536
Fit Width property of an **ImageView** (JavaFX) 797
five-pointed star 433
fixed text 32
 in a format string 28, 946
flag value 74
flags 946, 956
flash drive 475
flatMap method of interface **Stream** (Java SE 8) 578
float
 literal suffix F 532
 primitive type 31, 53, 898, 899
 primitive type promotions 126
Float class 510
floating-point constant 94
floating-point conversion specifiers 955
floating-point literal 53
 double by default 53
floating-point number 53, 73, 75, 76, 532, 948
 division 77
 double precision 53
 double primitive type 53
 float primitive type 53
 single precision 53
floor method of **Math** 120
flow of control 70, 76
flow of control in the if...else statement 63
FlowLayout class 342, 388, 389
 CENTER constant 391
 LEFT constant 391
 RIGHT constant 391
 setAlignment method 391
flush method
 of class **BufferedOutputStream** 504
Flyweight design pattern 996
focus 345, 805
focus for a GUI application 613, 628
FocusAdapter class 378
FocusListener interface 378
font
 manipulation 405
 name 414
font (cont.)
 size 414
 style 414
Font class 359, 403, 414
 BOLD constant 414,
 getFamily method 413, 416
 getName method 413, 414
 getSize method 413, 414
 getStyle method 413, 416
 isBold method 413, 416
 isItalic method 413, 416
 isPlain method 413, 416
 ITALIC constant 414
 PLAIN constant 414
font information 403
font manipulation 405
font metrics 416
 ascent 418
 descent 418
 height 418
 leading 418
Font property of a **Label** (JavaFX) 796
font style 358
FontMetrics class 403, 416
 getAscent method 417
 getDescent method 417
 getFontMetrics method 416
 getHeight method 417
 getLeading method 417
for repetition statement 62, 88, 89, 91, 92, 94, 95, 898
 activity diagram 92
 enhanced 163
 example 92
 header 90
 nested 154, 175
 nested enhanced for 175
forDigit method of class **Character** 461
forEach method of interface **IntStream** (Java SE 8) 556
forEach method of interface **Map** (Java SE 8) 573
forEach method of interface **Stream** (Java SE 8) 563
foreign key 737, 738
fork/join framework 730
format method
 of class **Formatter** 484, 962
 of class **String** 196, 962
format method of class **NumberFormat** 231, 722
format specifiers 28, 946
 %.2f for floating-point numbers
 with precision 78
 %% 953
 %B 952
 %b 952
 %b for boolean values 113
 %c 949
 %d 32, 946, 947
 %E 948
 %e 948
 %f 57, 948
 %G 948
 %g 948
 %H 953
 %h 952
 %n 953
format specifiers (cont.)
 %n (line separator) 29
 %o 947
 %S 949
 %s 29, 946, 949
 %X 947
 %x 947
format string 28, 946, 955
formatted output 952
 , (comma) formatting flag 96
 %f format specifier 57
 - (minus sign) formatting flag 95
 0 flag 154, 196
aligning decimal points in output 945
boolean values 113
 comma (,) formatting flag 96
 conversion character 946
 date and time compositions 950
 date and time conversion suffix characters 950
 dates 946
 exponential format 946
 field width 95, 946
 floating-point numbers 57
 grouping separator 96
 inserting literal characters 945
 integers in hexadecimal format 946
 integers in octal format 946
 left justification 945
 left justify 95
 minus sign (-) formatting flag 95
 precision 57, 946
 right justification 95, 945
 rounding 945
 times 946
Formatter class 477, 481, 945, 961
 close method 484
 format method 484, 962
 toString method 962
FormatterClosedException class 484
formatting
 display formatted data 28
Formatting date and time with conversion character t 951
Formatting output with class **Formatter** 961
fragile software 255
frame (in the UML) 850
Frame class 616
frequency method of **Collections** 519, 529
FROM SQL clause 739
fully qualified class name 48
fully qualified type name 224
Function functional interface (Java SE 8) 551, 565
 apply method 565
 identity method 580
function key 384
functional interface 548, 726
functional interface (Java SE 8) 302, 551
Functional Interfaces
 Supplier 726
Functional interfaces (Java SE 8)
 @FunctionalInterface annotation 582

- functional interfaces
 ActionListener 581
 ItemListener 581
 Functional interfaces (Java SE 8) 550, 551
 BiConsumer 573, 580
 BinaryOperator 551
 Consumer 551, 556, 569
 Function 551, 565
 IntFunction 723
 IntToDoubleFunction 723
 IntToLongFunction 723
 IntUnaryOperator 723
 Predicate 551, 569
 Supplier 551
 UnaryOperator 551
 functional programming 550
@FunctionalInterface annotation 582
 Future interface 726
 get method 726, 730
 fxid property of a JavaFX component 802
 FXML (FX Markup Language) 789
@FXML annotation 809, 810
 FXML markup 794
 FXMLLoader class (JavaFX) 801, 807, 810, 811
 load method 801, 807
- G**
- g command line option to javac 902
 game playing 128
 gaming console 8
 Gamma, Erich 993
 "Gang of Four" 993, 996, 997, 1005
 garbage collection 655
 garbage collector 216, 313, 317
 general class average problem 74
 general path 433
 generalities 267
 generalization in the UML 862
 GeneralPath class 403, 433
 closePath method 435
 lineTo method 435
 moveTo method 434
 generic class 189
 generics 509, 585
 ? (wildcard type argument) 607
 actual type arguments 589
 angle brackets (< and >) 588
 class 585, 594
 default upper bound (Object) of a type parameter 596
 diamond notation 514
 erasure 590
 interface 591
 method 585, 587, 593
 parameterized class 594
 parameterized type 594
 scope of a type parameter 596
 type parameter 588
 type parameter section 588
 type variable 588
 upper bound of a type parameter 592, 593
 upper bound of a wildcard 607
 wildcard type argument 607
- generics (cont.)
 wildcard without an upper bound 609
 wildcards 605, 607
 get method
 of class ArrayList<T> 191
 of class BitSet 984
 of interface Future 726
 of interface List<T> 513
 of interface Map 540
 get method 202, 209
 get method of class Paths 477, 478
 get method of interface Future 730
 getActionCommand method of class ActionEvent 349, 357
 getAscent method of class FontMetrics 417
 getAsDouble method of class OptionalDouble (Java SE 8) 557, 575
 getAutoCommit method of interface Connection 786
 getBlue method of class Color 407, 409
 getChars method
 of class String 439
 of class StringBuilder 454
 getClass method of class Object 343
 getClass method of Object 262, 287
 getClassName method of class StackTraceElement 324
 getClassName method of class UIManager.LookAndFeelInfo 632
 getClickCount method of class MouseEvent 380
 getColor method of class Color 407
 getColor method of class Graphics 407
 getColumnClass method of TableModel 755, 760
 getColumnClassName method of ResultSetMetaData 760
 getColumnCount method of ResultSetMetaData 753, 760
 getColumnCount method of TableModel 755, 760
 getColumnName method of ResultSetMetaData 760
 getColumnName method of TableModel 755, 760
 getColumnType method of ResultSetMetaData 753
 getConnection method of DriverManager 752
 getContentPane method of class JFrame 369
 getDescent method of class FontMetrics 417
 getFamily method of class Font 413, 416
 getFileName method of class StackTraceElement 324
 getFileName method of interface Path 478
 getFont method of class Graphics 414
 getFontMetrics method of class FontMetrics 416
 getFontMetrics method of class Graphics 417
 getGreen method of class Color 407, 409
 getHeight method of class FontMetrics 417
 getIcon method of class JLabel 343
 getInstalledLookAndFeel method of class UIManager 629
 getInstance method of Calendar 952
 getInt method of ResultSet 754
 getKeyChar method of class KeyEvent 387
 getKeyCode method of class KeyEvent 387
 getKeyModifiersText method of class KeyEvent 387
 getKeyText method of class KeyEvent 387
 getLastModifiedTime method of class Files 478
 getLeading method of class FontMetrics 417
 getLineNumber method of class StackTraceElement 324
 getMessage method of class Throwable 324
 getMethodName method of class StackTraceElement 324
 getMinimumSize method of class Component 616
 getModifiers method of class InputEvent 387
 getName method of class Class 262, 287
 getName method of class Font 413, 414
 getObject method of interface ResultSet 754, 760
 getPassword method of class JPasswordField 349
 getPercentInstance method of class NumberFormat 722, 809
 getPoint method of class MouseEvent 383
 getPreferredSize method of class Component 615
 getProperty method of class Properties 541
 getRed method of class Color 407, 409
 getResource method of class Class 343
 getRow method of interface ResultSet 761
 getRowCount method of interface TableModel 755, 760
 getSelectedFile method of class JFileChooser 501
 getSelectedIndex method of class JComboBox 367
 getSelectedIndex method of class JList 370
 getSelectedText method of class JTextComponent 400
 getSelectedValuesList method of class JList 372
 getSize method of class Font 413, 414
 getSource method of class EventObject 349

getStackTrace method of class `Throwable` 324
getStateChange method of class `ItemEvent` 367
getStyle method of class `Font` 413, 416
getText method 811
getText method of class `JLabel` 343
getText method of class `JTextComponent` 625
getText method of class `TextInputControl` 811
getCurrencyInstance method of class `NumberFormat` 231
getValueFormat method of class `JSlider` 616
getValueAt method of interface `TableModel` 755, 760
getX method of class `MouseEvent` 377
getY method of class `MouseEvent` 377
GIF (Graphics Interchange Format) 343
 GitHub 7
 glass pane 369
 glyph 939
 Good Programming Practices xxxi
 Google Play 8
 Gosling, James 8
goto elimination 60
goto statement 60
 gradient 431
GradientPaint class 403, 431
 Grand, Mark 1005
 graph information 153
 graphical user interface (GUI) 301, 333, 788
 design tool 388
 graphics 381
Graphics class 382, 403, 405, 428
 clearRect method 419
 draw3DRect method 419, 422
 drawArc method 422
 drawLine method 419
 drawOval method 419, 422
 drawPolygon method 425, 427
 drawPolyline method 425, 427
 drawRect method 419, 432
 drawRoundRect method 420
 drawString method 409
 fill3DRect method 419, 422
 fillArc method 422
 fillOval method 383, 419, 422
 fillPolygon method 425, 428
 fillRect method 407, 419, 432
 fillRoundRect method 420
 getColor method 407
 getFont method 414, 414
 getFontMetrics method 417
 setColor method 432
 setFont method 414
 graphics context 405
 graphics in a platform-independent manner 405
 Graphics Interchange Format (GIF) 343
Graphics2D class 403, 428, 432, 435
 draw method 431
 fill method 431, 432, 435
 rotate method 435
 setPaint method 431
 setStroke method 431
 translate method 435

greedy quantifier 469
 grid 395
 grid for `GridBagLayout` layout manager 642
GridBagConstraints class 643, 648, 649
 anchor field 643
 BOTH constant 644
 CENTER constant 643
 EAST constant 643
 gridheight field 644
 gridwidth field 644
 gridx field 644
 gridy field 644
 HORIZONTAL constant 644
 instance variables 643
 NONE constant 644
 NORTH constant 643
 NORTHEAST constant 643
 NORTHWEST constant 643
 RELATIVE constant 649
 REMAINDER constant 649
 SOUTH constant 643
 SOUTHEAST constant 643
 SOUTHWEST constant 643
 VERTICAL constant 644
 weightx field 644
 weighty field 644
 WEST constant 643
GridBagConstraints constants
 RELATIVE and REMAINDER 649
GridBagLayout class 639, 642, 644, 649
 setConstraints method 649
GridBagLayout layout manager 645
gridheight field of class `GridBagConstraints` 644
GridLayout class 388, 395
GridLayout containing six buttons 395
GridPane class (JavaFX) 799, 799
 add rows or columns 803
 Hgap property 805
 Vgap property 805
gridwidth field of class `GridBagConstraints` 644
gridx field of class `GridBagConstraints` 644
gridy field of class `GridBagConstraints` 644
GROUP BY 739
group method of class `Matcher` 472
 grouping separator (formatted output) 96
groupingBy method of class `Collectors` (Java SE 8) 573, 574, 578, 580
 guard condition in the UML 62, 838
 Guarded Suspension design pattern 994, 1005
 guarding code with a lock 664
GUI (Graphical User Interface) 301
 component 333
 design tool 388
GUI (Graphical User Interface)
 component 788
 ImageView (JavaFX) 791
 Label (JavaFX) 791
 Label class (JavaFX) 791
 naming convention 802
 Slider (JavaFX) 798, 800

GUI (Graphical User Interface)
 component (cont.)
 TextField (JavaFX) 800
 TextField class (JavaFX) 800
 guillemets (« and ») 53

H

H conversion character 952
 h conversion character 952
 handle an exception 309
has-a relationship 210, 236, 828
 hash table 534, 538
hashCode method of `Object` 262
 hashing 537
HashMap class 537
 keySet method 541
HashSet class 534
Hashtable class 537, 538, 1011
 hash-table collisions 538
hasNext method
 of class `Scanner` 101, 484
 of interface `Iterator` 514, 517
hasPrevious method of `ListIterator` 517
headSet method of class `TreeSet` 535
 heavyweight components 339
 height 416
 height of a rectangle in pixels 407
 Helm, Richard 993
 Help link in API 916
helpdoc.html generated by javadoc 935
 helper method 197
 hexadecimal (base 16) number system 964
 hexadecimal integer 946
Hgap property of a `GridPane` 805
 hidden fields 138
 hide a dialog 336
 hide implementation details 199
HIDE_ON_CLOSE constant of interface `WindowConstants` 616
Hierarchy window in NetBeans 794, 795
 Hiragana block 942
 hold a lock 664
 hollow diamonds (representing aggregation) in the UML 828
 Home key 384
HORIZONTAL constant of class `GridBagConstraints` 644
 horizontal coordinate 403
 horizontal gap space 393
 horizontal glue 642
 horizontal `JSlider` component 613
 horizontal scrollbar policy 401
 horizontal tab 28
HORIZONTAL_SCROLLBAR_ALWAYS
 constant of class `JScrollPane` 401
HORIZONTAL_SCROLLBAR_AS_NEEDED
 constant of class `JScrollPane` 401
HORIZONTAL_SCROLLBAR_NEVER
 constant of class `JScrollPane` 401
 hot spots in bytecode 11
HourlyEmployee class derived from `Employee` 278
 hue 412

I

I/O performance enhancement 504
 IBM Corporation 938
 icon 337
Icon interface 343
 IDE (integrated development environment) 9
 identifier 23, 31
 identifiers
 camel case naming 40
 identity column 735, 770
IDENTITY keyword (SQL) 735
identity method of functional interface *Function* (Java SE 8) 580
 identity value in a reduction (Java SE 8) 558
 IEEE 754 (grouper.ieee.org/groups/754/) 899
 IEEE 754 floating point 899
if single-selection statement 34, 61, 62, 63, 98, 898
 activity diagram 63
if...else double-selection statement 61, 63, 64, 75, 98
 activity diagram 64
 ignoring array element zero 157
IllegalArgumentException class 196
IllegalMonitorStateException class 684, 699
IllegalStateException class 487
Image property of a *ImageView* 796, 797
ImageIcon class 343
ImageView class (JavaFX) 791
 Fit Width property 797
 Image property 796, 797
 immutability 550
 immutable 439
 immutable data 664
 immutable object 219
 immutable *String* object 439
 implement an interface 266, 290, 298
 implementation-dependent code 199
 implementation of a function 276
 implementation phase 867
 implementation process 840, 857
 implements 6
 implements 898
 implements keyword 290, 295
 implements multiple interfaces 374
 implicit conversion 77
import declaration 30, 31, 48, 898
 in parallel 654
 increment 93
 a control variable 87
 expression 109
 of a **for** statement 91
 operator, ++ 82
 increment a control variable 87
 increment and decrement operators 82
 indefinite postponement 659, 703
 indefinite repetition 74
 indentation 63, 65
 index (subscript) 146, 155
 Index link in API 916
 index of a *JComboBox* 366
 index zero 146

I generated by javadoc 934
Index_CD-all.html generated by javadoc 935
indexOf method of class *ArrayList<T>* 189
indexOf method of class *String* 445
IndexOutOfBoundsException class 526
IndexOutOfBoundsException class 158
 indirect superclass 235, 237
InetAddress class 1007
 infer a type with the diamond (<>) notation 192
 infer parameter types in a lambda 556
 infinite loop 70, 77, 91
 infinite recursion 260
 infinite stream (Java SE 8) 580
 infinity symbol 738
 information hiding 5, 48
 information tier 1009
 inheritance 5, 233, 235, 862, 865, 866, 867
 examples 236
 extends keyword 239, 250
 hierarchy 236, 272
 hierarchy for university
 CommunityMembers 237
 multiple 235
 single 235
 initial state in the UML 61, 835, 836
 initial value of an attribute 833
 initial value of control variable 87
 initialize a controller's instance variables 807
 initialize a variable in a declaration 31
initialize method of a JavaFX controller class 811
 initializer list 149
 initializing two-dimensional arrays in declarations 174
 initiate an action 618
 inlining method calls 206
 inner class 347, 360, 382, 624
 anonymous 366, 812
 object of 360
 relationship between an inner class and its top-level class 360
INNER JOIN SQL clause 739, 744
 innermost set of brackets 156
 input data from the keyboard 37
 input dialog 335
 input/output operation 61
 input/output package 127
InputEvent class 373, 380, 384
 getModifiers method 387
 isAltDown method 380, 387
 isControlDown method 387
 isMetaDown method 380, 387
 isShiftDown method 387
InputMismatchException class 308, 311
InputStream class 493, 503, 544
InputStreamReader class 506
 insert method of class *StringBuilder* 457
INSERT SQL statement 739, 745

inserting literal characters in the output 945
 insertion point 188, 528
Inspector window in NetBeans 795
 instance 4
 instance (non-static) method 217
 instance method reference (Java SE 8) 565
 instance methods 124
 instance variable 5, 41, 54, 120
 instance variables 41
instanceof operator 286, 898
Instant class
 now method 722
Instant class (Java SE 8) 729
int primitive type 31, 75, 81, 98, 898, 899
 promotions 126
IntBinaryOperator functional interface (Java SE 8) 558
 applyAsInt method 558
IntConsumer functional interface (Java SE 8) 556
 accept method 556
integer 29
 array 150
 division 73
 quotient 33
 value 31
Integer class 184, 337, 510, 606
 parseInt method 184, 337
 toBinaryString method 976
 integer conversion characters 946
 integer division 33
 integers
 suffix L 532
 integral expression 104
 integrated development environment (IDE) 9
 intelligent consumer electronic device 8
 IntelliJ IDEA (www.jetbrains.com) 9
 interaction diagram in the UML 848
 interaction overview diagram in the UML 991
 interactions among objects 845, 849
 interest rate 94
 interface 6, 266, 291, 299, 753
 declaration 290
 implementing more than one at a time 374
 tagging interface 493
interface keyword 290, 898
 Interfaces 290
 ActionListener 348, 353
 AutoCloseable 216, 330, 752
 BiConsumer functional interface (Java SE 8) 573, 580
 BinaryOperator functional interface (Java SE 8) 551
 BlockingQueue 680
 CachedRowSet 767
 Callable 726
 CallableStatement 785
 ChangeListener 616
 ChangeListener (JavaFX) 801, 808, 812
 CharSequence 471
 Collection 509, 510, 519

- Interfaces (cont.)
- Collector functional interface
 - (Java SE 8) 563
 - Comparable 300, 443, **520, 590**
 - Comparator 520, 521, 566
 - ComponentListener 378
 - Condition 699, 700
 - Connection 752, 754, 759
 - Consumer functional interface (Java SE 8) 551, 556, 569
 - ContainerListener 378
 - DataInput 504
 - DataOutput 504
 - default methods (Java SE 8) 301, 301
 - DirectoryStream 477
 - DoubleStream functional interface (Java SE 8) 554
 - EventHandler<ActionEvent> (JavaFX) 811
 - Executor 659
 - ExecutorService 660, 726
 - FocusListener 378
 - Function functional interface (Java SE 8) 551, 565
 - Future 726
 - Icon 343
 - IntBinaryOperator functional interface (Java SE 8) 558
 - IntConsumer functional interface (Java SE 8) 556
 - IntFunction functional interface (Java SE 8) 723
 - IntPredicate functional interface (Java SE 8) 559
 - IntStream functional interface (Java SE 8) 554
 - IntToDoubleFunction functional interface (Java SE 8) 723
 - IntToLongFunction functional interface (Java SE 8) 723
 - IntUnaryOperator functional interface (Java SE 8) 560, 723
 - ItemListener 360, 624
 - Iterator 510
 - JdbcRowSet 767
 - KeyListener 353, 378, 384
 - LayoutManager 387, 391
 - LayoutManager2 391
 - List 509, 517
 - ListIterator 511
 - ListSelectionListener 369
 - Lock 698
 - LongStream functional interface (Java SE 8) 554
 - Map 509, 537
 - Map.Entry 578
 - MouseInputListener 372, 377
 - MouseListener 353, 372, 378, 628
 - MouseMotionListener 353, 372, 377, 378
 - MouseWheelListener 374
 - ObjectInput 492
 - ObjectOutput 492
 - ObservableValue (JavaFX) 809
 - Path 477
 - Predicate functional interface (Java SE 8) 551, 569
- Interfaces (cont.)
- PreparedStatement 785
 - PropertyChangeListener 720
 - Queue 509, 533, 680
 - ResultSet 753
 - ResultSetMetaData 753
 - RowSet 767
 - Runnable 659, 301
 - Serializable 301, 493
 - Set 509, 534
 - SortedMap 537
 - SortedSet 535
 - Statement 754
 - static methods (Java SE 8) 302
 - Stream (Java SE 8) **552, 562**
 - Supplier 726, 729
 - Supplier functional interface (Java SE 8) 551
 - SwingConstants 343, 616
 - TableModel 754
 - ToDoubleFunction functional interface (Java SE 8) 575
 - UnaryOperator functional interface (Java SE 8) 551
 - WindowConstants 616
 - WindowListener 377, 378, 617, 766
- intermediate operation 559
- intermediate operations
- stateful 560
 - stateless 560
- intermediate stream operations (Java SE 8)
- filter method of interface IntStream **559**
 - filter method of interface Stream **563, 566**
 - flatMap method of interface Stream **578**
 - map method of interface IntStream **560**
 - map method of interface Stream **565**
 - sorted method of interface IntStream **559**
 - sorted method of interface Stream **563, 566**
- internal frame
- closable 636
 - maximizable 636
 - minimizable 636
 - resizable 636
- internal iteration 550
- internationalization 231, 800
- Internet domain name in reverse order 224
- Interpreter design pattern 997
- interrupt method of class Thread **661**
- InterruptedException class **661**
- IntFunction functional interface (Java SE 8) 723
- IntPredicate functional interface (Java SE 8) 559
- test method 559, **560**
- intrinsic lock 664
- ints method of class SecureRandom (Java SE 8) **580**
- IntStream interface (Java SE 8) 554
- average method 557
 - boxed method 580
- IntStream interface (Java SE 8) (cont.)
- count method 557
 - filter method 559
 - forEach method 556
 - map method 560
 - max method 557
 - min method 557
 - of method 556
 - range method 561
 - rangeClosed method 561
 - reduce method 557
 - sorted method 559
 - sum method 557
- IntToDoubleFunction functional interface (Java SE 8) 723
- IntToLongFunction functional interface (Java SE 8) 723
- IntUnaryOperator functional interface (Java SE 8) **560, 723**
- IOException class 497
- is-a* relationship 236, 267
- isAbsolute method of interface Path 478
- isActionKey method of class KeyEvent 387
- isAltDown method of class InputEvent 380, 387
- isBold method of class Font 413, **416**
- isCancelled method of class SwingWorker 716
- isControlDown method of class InputEvent 387
- isDefined method of class Character 460
- isDigit method of class Character 460
- isDirectory method of class Files 478
- isEmpty method
- ArrayList 210
 - Map 541
 - Stack 533
- isItalic method of class Font 413, **416**
- isJavaIdentifierPart method of class Character 460
- isJavaIdentifierStart method of class Character 460
- isLetter method of class Character 460
- isLetterOrDigit method of class Character 460
- isLowerCase method of class Character 460
- isMetaDown method of class InputEvent 380, 387
- isPlain method of class Font 413, **416**
- isPopupTrigger method of class MouseEvent 628
- isSelected method
- AbstractButton 625
 - JCheckBox 360
- isShiftDown method of class InputEvent 387
- isUpperCase method of class Character 460
- ITALIC constant of class Font 414
- ItemEvent class 360, 363
- getStateChange method 367

I
ItemListener interface 360, 624
 itemStateChanged method 360, 625
itemStateChanged method of interface *ItemListener* 360, 625
iteration 72
 of a *loop* 87, 109
 iteration (looping)
 of a *for loop* 156
iteration statements 62
iterative model 820
iterator 508
 fail fast 514
Iterator design pattern 994, 997, 1011
Iterator interface 510
 hasNext method 514
 next method 514
 remove method 514
iterator method of *Collection* 514

J

Jacopini, G. 60
JApplet class 617
java .time package 199
Java 2D API 403, 428
Java 2D shapes 428
Java 2D Shapes package 127
Java Abstract Window Toolkit Event package 127
Java API 118
 overview 128
Java API documentation 128
 download 32
 java.sun.com/javase/6/docs/api/ 32, 915
 online 32
Java Application Programming Interface (Java API) 8, 30, 118, 127
Java class library 8, 30, 118
java command 10, 13, 21
Java compiler 10
Java Concurrency Package 127
Java Database Connectivity (JDBC) 733
Java DB 733, 747
 embedded 747
Java DB Developer's Guide 735
Java debugger 901
Java development environment 9, 10, 11, 12
Java Development Kit (JDK) 24
Java Enterprise Edition (Java EE) 3
 .java extension 9
 .java file name extension 40
Java fonts
 Dialog 414
 DialogInput 414
 Monospaced 414
 SansSerif 414
 Serif 414
Java HotSpot compiler 11
Java Input/Output Package 127
java interpreter 25
Java Keywords 898
Java Language Package 127
Java Micro Edition (Java ME) 3
Java programming language 7
Java SE 7 (Java Standard Edition 7) 3

Java SE 8 146, 164, 172, 188, 199, 723, 726
@FunctionalInterface
 annotation 582
 anonymous owner classes with lambdas 367
Arrays method *parallelSort* 188
BinaryOperator functional interface 551
Collector functional interface 563
Collectors class 563
CompletableFuture class 726
Consumer functional interface 551, 556, 569
Date/Time API 188, 199
Date/Time API 128
default interface methods 301
default method in an interface 550, 581
default method of an interface 548
default methods in interfaces 301
doubles method of class *SecureRandom* 580
effectively final 366
Function functional interface 551, 565
 functional interface 302
 functional interfaces 551
@FunctionalInterface
 annotation 582
 implementing event listeners with lambdas 350, 612
Instant class 729
IntBinaryOperator functional interface 558
IntConsumer functional interface 556
IntFunction functional interface 723
IntPredicate functional interface 559
ints method of class *SecureRandom* 580
IntToDoubleFunction functional interface 723
IntToLongFunction functional interface 723
IntUnaryOperator functional interface 560, 723
java.util.function package 550, 556
java.util.stream package 554
lambda 302
 lambdas and streams with regular expressions 473
lines method of class *Files* 577
longs method of class *SecureRandom* 580
Optional 570
OptionalDouble class 557
Predicate functional interface 551, 563, 566, 569
reversed method of interface *Comparator* 566
static interface methods 302
static method in an interface 550, 581
static method of an interface 548
Java SE 8 (cont.)
Stream interface 562
Supplier functional interface 551
Supplier interface 726, 729
ToDoubleFunction functional interface 575
UnaryOperator functional interface 551
Java SE 8 (Java Standard Edition 8) 3
Java SE 8 Development Kit (JDK) 9
Java Security Package 127
Java Standard Edition (Java SE) 7 3
 8 3
Java Standard Edition 8 (Java SE 8) 3
Java Swing Event Package 127
Java Swing GUI Components Package 127
Java Utilities Package 127
Java Virtual Machine (JVM) 10, 21, 23
JAVA_HOME environment variable 748
java.awt package 338, 405, 406, 425, 428, 616, 628
java.awt.color package 428
java.awt.event package 127, 350, 352, 377, 387
java.awt.font package 428
java.awt.geom package 127, 428
java.awt.image package 428
java.awt.image.renderable package 428
java.awt.print package 428
java.beans package 720
java.io package 127, 476
java.lang package 31, 119, 127, 239, 261, 437, 659, 1010
 imported in every Java program 32
java.math package 78, 230, 808
java.nio.file package 475, 476, 477, 577
java.security package 128
java.sql package 127, 750, 753
java.text package 230, 800, 808
java.time package 128
java.util package 30, 127, 188, 509, 531, 951
 Calendar class 951
 Date class 951
java.util.concurrent package 127, 660, 680, 705, 726
java.util.concurrent.locks package 698, 699
java.util.function package (Java SE 8) 550, 556
java.util.prefs package 541
java.util.regex package 437
java.util.stream package (Java SE 8) 554
Java™ Language Specification (java.sun.com/docs/books/jls/) 33
Java2D API 428
javac compiler 10, 24
javacdeprecation flag 30
Javadoc comment 22
javadoc options
 -author 933
 -d 933
 -link 933

javadoc tag 924
javadoc tags
 {@link} 932
 @author 929
 @deprecated 932
 @param 930
 @return 930
 @see 929
 @since 932
 @throws 930
 @version 932
javadoc utility program 22, 924
JavaFX 333, 788
 @FXML annotation 809, 810
 ActionEvent class 809, 811
 alignment in a VBox 796
 Application class 799
 Cascading Style Sheets (CSS) 789
 ChangeListener interface 801,
 808, 812
 controller class 791, 801
 DoubleProperty class 812
 EventHander<ActionEvent>
 interface 811
 fx:id property 802
 FXML (FX Markup Language) 789
 FXMLLoader class 801, 807
 GridPane class 799, 799
 ImageView class 791
 Label class 791
 Max Width property 805
 node 790
 Node class 790
 padding 805
 Padding property 805
 Parent class 802, 807
 Pref Height property of a
 component 796
 Pref Width property 804
 Pref Width property of a component
 796
 preferred size 796
 register event handlers 807
 scene 790
 Scene class 790, 799, 807, 808
 scene graph 790
 Slider class 798, 800
 stage 790
 Stage class 790, 799, 807, 808
 TextField class 800
 Vbox class 795
JavaFX FXML Application NetBeans
 project 791
JavaFX Scene Builder 789, 791
 change the default layout 795, 803
JavaFX Script 788
javafx.application.Application
 package 799
javafx.beans.value package 808,
 812
javafx.event package 809
javafx.fxml package 809
javafx.scene package 790, 802, 807
javafx.scene.control package 800,
 809
javafx.scene.layout package 795,
 799
javafx.stage package 790
JavaScript 788
javax.sql.rowset package 767
javax.swing package 127, 333, 335,
 343, 352, 354, 399, 410, 616, 629,
 635
javax.swing.event package 127, 351,
 369, 377, 616
javax.swing.table package 755, 766
JButton class 338, 354, 357, 393
JCheckBox buttons and item events 358
JCheckBox class 338, 357
 isSelected method 360
JCheckBoxMenuItem class 617, 618,
 624
JColorChooser class 410, 412
 showDialog method 411
JComboBox class 338, 363, 644
 getSelectedIndex method 367
 setMaximumRowCount method 366
JComboBox that displays a list of image
 names 364
JComponent class 339, 340, 342, 352,
 364, 367, 381, 397, 403, 405, 1001
 paintComponent method 381,
 403, 613, 615
 repaint method 406
 setForeground method 624
 setOpaque method 381, 383
 setToolTipText method 342
jdb command 902
JDBC
 API 733, 750, 785
 driver 733
JDBC Package 127
jdbctery:books 752
JdbcRowSet interface 767
 close method 769
 execute method 769
 setCommand method 769
 setPassword method 769
 setUrl method 767
 setUsername method 769
JDesktopPane class 633
JDesktopPane documentation 636
JDialog class 623
JDK 9, 24
JFileChooser class 500
 CANCEL_OPTION constant 501
 FILES_AND_DIRECTORIES
 constant 500
 FILES_ONLY constant 500
 getSelectedFile method 501
 setFileSelectionMode method
 500
 showOpenDialog method 500
JFrame class 616
 add method 342
 EXIT_ON_CLOSE 344
 getContentPane method 369
 setDefaultCloseOperation
 method 344, 616
 setJMenuBar method 617, 624
 setSize method 344
 setVisible method 344
JFrame.EXIT_ON_CLOSE 344
JInternalFrame class 633, 635
JInternalFrame documentation 636
JLabel class 338, 340
 getIcon method 343
 getText method 343
JLabel class (cont.)
 setHorizontalAlignment
 method 343
 setHorizontalTextPosition
 method 343
 setIcon method 343
 setText method 343
 setVerticalAlignment method
 343
 setVerticalTextPosition
 method 343
JList class 338, 367
 addListSelectionListener
 method 369
 getSelectedIndex method 370
 getSelectedValuesList method
 372
 setFixedCellHeight method 372
 setFixedCellWidth method 372
 setListData method 372
 setSelectionMode method 369
 setVisibleRowCount method 369
JMenu class 617, 624, 635
 add method 623
 addSeparator method 624
JMenuBar class 617, 624, 635
 add method 624
JMenuItem class 618, 635
JMenus and mnemonics 618
Johnson, Ralph 993
JOIN_ROUND constant of class
 BasicStroke 433
joining database tables 738, 744
Joint Photographic Experts Group
 (JPEG) 343
JOptionPane class 335, 336
 constants for message dialogs 338
 PLAIN_MESSAGE constant 337
 showInputDialog method 336
 showMessageDialog method 337
JOptionPane constants for message
 dialogs
 JOptionPane.ERROR_MESSAGE
 338
 JOptionPane.INFORMATION_MESSAGE
 SAGE 338
 JOptionPane.PLAIN_MESSAGE
 338
 JOptionPane.QUESTION_MESSAGE
 E 338
 JOptionPane.WARNING_MESSAGE
 338
JPanel class 338, 381, 388, 397, 613,
 1001
JPasswordField class 344, 349
 getPassword method 349
JPEG (Joint Photographic Experts
 Group) 343
JPopupMenu class 625
 show method 628
JProgressBar class 717
JRadioButton class 357, 360, 363
JRadioButtonMenuItem class 617,
 618, 625
JScrollPane class 369, 372, 400, 401
 HORIZONTAL_SCROLLBAR_ALWAYS
 constant 401
 HORIZONTAL_SCROLLBAR_AS_NEEDED
 constant 401
DED constant 401

JScrollPane class (cont.)
 HORIZONTAL_SCROLLBAR_NEVER
 constant 401
 setHorizontalScrollBarPolicy
 method 401
 setVerticalScrollBarPolicy
 method 401
 VERTICAL_SCROLLBAR_ALWAYS
 constant 401
 VERTICAL_SCROLLBAR_AS_NEEDED
 D constant 401
 VERTICAL_SCROLLBAR_NEVER
 constant 401
 JScrollPane scrollbar policies 401
 JSlider class 612, 613, 616
 block increment 613
 getValue method 616
 major tick marks 612
 minor tick marks 612
 setInverted method 613
 setMajorTickSpacing method
 616
 setPaintTicks method 616
 snap-to ticks 612
 thumb 612
 tick marks 612
 JTabbedPane class 636, 642
 addTab method 638
 SCROLL_TAB_LAYOUT constant 642
 TOP constant 642
 JTable class 754
 RowFilter 766
 setRowFilter method 766
 setRowSorter method 766
 TableRowSorter 766
 JTextArea class 387, 398, 400, 644,
 647
 setLineWrap method 401
 JTextComponent class 344, 347, 398,
 400
 getSelectedText method 400
 getText method 625
 setDisabledTextColor method
 387
 setEditable method 347
 setText method 400
 JTextField class 338, 344, 348, 352,
 398
 addActionListener method 348
 JTextField and JPasswordField
 345
 JToggleButton class 357
 just-in-time compilation 11
 just-in-time (JIT) compiler 11

K

key constant 387
 key event 353, 384
 Key event handling 384
 key-value pair 538
 KeyAdapter class 378
 keyboard 29, 333, 788
 KeyEvent class 353, 384
 getKeyChar method 387
 getKeyCode method 387
 getKeyModifiersText method 387
 getKeyText method 387
 isActionKey method 387

L

label 340, 988
 Label class (JavaFX) 791
 Font property 796
 Text property 796

keyListener interface 353, 378, 384
 keyPressed method 384, 387
 keyReleased method 384
 keyTyped method 384
 Keypad class (ATM case study) 823, 826,
 827, 828, 839, 846, 847, 848, 850,
 859, 862, 893
 keyPressed method of interface
 KeyListener 384, 387
 keyReleased method of interface
 KeyListener 384
 keySet method
 of class HashMap 541
 of class Properties 544
 keyTyped method of interface
 KeyListener 384
 keyword 22, 62
 Keywords
 abstract 271
 boolean 66, 906
 break 102
 case 102
 catch 311
 char 31
 class 22, 40
 continue 108
 default 102
 do 62, 97
 double 31, 53
 else 62
 enum 136
 extends 239, 250
 false 66, 898
 final 104, 120, 151, 664
 finally 311
 float 31, 53
 for 62, 88
 if 62
 implements 290
 import 30
 instanceof 286
 int 31
 interface 290
 new 31, 44, 147, 148
 null 50, 147, 898
 private 41, 199, 209
 public 22, 40, 41, 122, 199
 reserved but not used by Java 898
 return 40, 42, 125
 static 95, 119
 super 238, 261
 switch 62
 synchronized 664
 table of Keywords and reserved words
 898
 this 42, 200, 217
 throw 321
 true 66, 898
 try 310
 void 23, 41
 while 62, 97
 Koenig, Andrew 305

label in a switch 102
 labeled block 988
 labeled break statement 987
 exiting a nested for statement 987
 labeled continue statement 988
 terminating a single iteration of a
 labeled-for statement 989
 labeled statement 987, 988
 labels for tick marks 612
 lambda (Java SE 8) 302
 lambda expression
 composing 560
 type of 551
 lambda expression (Java SE 8) 551
 lambda expressions
 arrow token (->) 551
 event handler 581
 method references 552
 parameter list 551
 target type 556
 with an empty parameter list 552
 lambdas
 implementing an event handler 813
 lambda expressions
 statement block 551
 type inference 556
 LAMP 17
 language package 127
 last-in, first-out (LIFO) order 599
 last method of ResultSet 761
 last method of SortedSet 536
 lastIndexOf method of class String
 445
 late binding 286
 launch method of class Application
 (JavaFX) 799, 807
 Layers architecture pattern 994, 1010
 layout containers (JavaFX) 790
 layout manager 342, 377, 387, 396
 BorderLayout 377
 FlowLayout 342
 GridLayout 395
 layoutContainer method of interface
 LayoutManager 391
 LayoutManager interface 387, 391
 layoutContainer method 391
 LayoutManager2 interface 391
 Layouts
 GridPane 799
 Vbox 795
 lazy 552
 lazy quantifier 469
 lazy stream operation (Java SE 8) 559,
 560
 Lea, Doug 1005
 leading 416
 leaf 1000
 left brace, { 23, 30
 LEFT constant of class FlowLayout 391
 left justification 945
 left justified 63, 95, 343, 389
 left-mouse-button click 380
 left shift (<<) 973, 974, 980, 981
 Left, center and right mouse-button clicks
 378
 length field of an array 147
 length instance variable of an array 147
 length method of class String 439

length method of class **StringBuilder** 452
 level of indentation 63
 levels of nesting 988
 lexical scope 556
 lexicographical comparison 442, 443
Library window in NetBeans 795, 796
 life cycle of a thread 656, 658
 lifeline of an object in a UML sequence diagram 850
 LIFO (last-in, first-out) 599
 lightweight GUI component 339
LIKE operator (SQL) 740
LIKE SQL clause 741, 743
 line 403, 418, 427
 line join 431
 line wrapping 401
Line2D class 403, 432
Line2D.Double class 428
LinearGradientPaint class 431
LineNumberReader class 506
lines method of class **Files** (Java SE 8) 577
lineTo method of class **GeneralPath** 435
{**@link**} javadoc tag 932
-link option 933
linked list 1011
LinkedBlockingDeque class 706
LinkedBlockingQueue class 706
LinkedList class 511, 527
 add method 518
 addFirst method 519
 addLast method 518
LinkedTransferQueue class 706
Linux 24, 483
Linux operating system 7
list 366
List interface 509, 517, 520, 525
 add method 513, 516
 addAll method 516
 clear method 517
 get method 513
 listIterator method 517
 size method 513, 517
 subList method 517
 toArray method 518
list method of **Properties** 543
listen for events 348
ListIterator interface 511
 hasPrevious method 517
 previous method 517
 set method 517
listIterator method of interface **List** 517
ListSelectionEvent class 367
ListSelectionListener interface 369
 valueChanged method 369
ListSelectionMode class 369
 MULTIPLE_INTERVAL_SELECTION constant 369, 372
 SINGLE_INTERVAL_SELECTION constant 369, 370, 372
 SINGLE_SELECTION constant 369
literals
 floating point 53
load factor 538
load method of class **FXMLLoader** (JavaFX) 801, 807
load method of **Properties** 544
loading 10
local variable 42, 72, 138
locale-specific currency Strings 231
locale-specific **String** 230
localization 339, 937
lock
 acquire 664
 hold 664
 release 664
lock an object 686, 687
Lock interface 698
 lock method 698, 703
 newCondition method 699, 700
 unlock method 698, 703
lock method of interface **Lock** 698, 703
log method of **Math** 120
logarithm 120
logic error 9, 32, 901
logical complement operator, ! 112
logical input operations 505
logical negation, ! 112
logical negation, or logical NOT () operator truth table 113
logical operators 110, 112
logical output operations 504
long
 literal suffix L 532
Long class 510
long keyword 898, 899
long promotions 126
longs method of class **SecureRandom** (Java SE 8) 580
LongStream interface (Java SE 8) 554
look-and-feel 338, 339, 387, 628
 Nimbus 335
Look-and-Feel Observations
 overview xxxii
Look-and-feel of a Swing-based GUI 629
look-and-feel of an application 338
look-and-feel 335
LookAndFeelInfo nested class of class **UIManager** 629
lookingAt method of class **Matcher** 471
loop 70, 72
 body 97
 continuation condition 62
 counter 87
 infinite 70, 77
 statement 62
loop-continuation condition 87, 87, 89, 91, 93, 97, 98, 109
looping 72
lowercase letter 22
lowered rectangle 422

M

m-by-n array 173
Mac OS X 24, 483
Macintosh look-and-feel 629
main method 30, 47
main thread 663
major tick marks of class **JSlider** 612
make your point (game of craps) 133
making decisions 37
many-to-many relationship 738
many-to-one mapping 537
many-to-one relationship in the UML 829
map elements of a stream (Java SE 8) 560
Map interface 509, 537
 containsKey method 540
 forEach method (Java SE 8) 573
 get method 540
 isEmpty method 541
 put method 540
 size method 541
map method of interface **IntStream** (Java SE 8) 560
map method of interface **Stream** (Java SE 8) 565
Map.Entry interface 578
mapToDouble method of interface **Stream** (Java SE 8) 574
marker interfaces 291
mask 974
Matcher class 437, 471
 find method 471
 group method 472
 lookingAt method 471
 matches method 471
 replaceAll method 471
 replaceFirst method 471
matcher method of class **Pattern** 471
matches method of class **Matcher** 471
matches method of class **Pattern** 471
matches method of class **String** 464
matching catch block 311
Math class 95, 119
 abs method 119
 ceil method 120
 cos method 120
 E constant 120
 exp method 120
 floor method 120
 log method 120
 max method 120
 min method 120
 PI constant 120
 pow method 95, 96, 119, 120
 sqrt method 119, 120, 125
 tan method 120
MathContext class 232
max method of **Collections** 519, 526
max method of interface **IntStream** (Java SE 8) 557
max method of **Math** 120
Max property of a **Slider** (JavaFX) 806
Max Width property of a JavaFX control 805
maximize a window 340, 635
maximized internal frame 636
MBCS (multi-byte character set) 940
MDI (Multiple Document Interface) 612, 633
Mediator design pattern 997
Memento design pattern 994, 997, 998
memento object 998
memory buffer 505
memory leak 216, 317
memory-space/execution-time trade-off 538
memory utilization 538
menu 334, 398, 617, 618

- menu bar 334, 617
 menu item 618, 623
 merge in the UML 838
 merge records from tables 743
 merge symbol in the UML 70
 message dialog 335, 337
 types 337
 message in the UML 845, 848, 849, 850
 message passing in the UML 850
Meta key 380
 metadata 753
 metal look-and-feel 612, 629
 method 4, 23, 857
 local variable 42
 parameter list 42
 signature 142
 static 95
 method call 5, 122
 method declaration 122
 Method Detail section in API 922
 method header 41
 method names
 came case naming 40
 method overloading 140
 method parameter list 182
 method reference 565
 method reference (Java SE 8) 565
 method references 552
 Method Summary section in API 920
 methods implicitly final 289
 Microsoft 938
 Microsoft SQL Server 733
 Microsoft Windows 101, 616, 628
 Microsoft Windows-style look-and-feel 629
 middle mouse button 380
 middle tier 1009
`min` method of *Collections* 519, 526
`min` method of interface *IntStream* (Java SE 8) 557
`min` method of *Math* 120
 minimize a window 340, 617, 635
 minimize internal frame 635
 minor tick marks of class *JSlider* 612
 minus sign (-) formatting flag 95
 minus sign (-) indicating private visibility
 in the UML 857
 mnemonic 339, 618, 622, 624
 mobile application 3
 modal dialog 337, 412
 modal dialog box 623
 model (in MVC architecture) 801, 1008
 model of a software system 827, 834, 864
 Model-View-Controller (MVC) 801,
 994, 1002, 1008
 modifier key 387
 monetary calculations 96, 230
 monitor 664
 monitor lock 664
Monospaced Java font 414
 Motif-style (UNIX) look-and-feel 612,
 629
 mouse 333, 788
 mouse-button click 380
 mouse click 378
 mouse event 353, 372, 628
 handling 373
 mouse wheel 374
MouseAdapter class 377, 378
mouseClicked method of interface
 MouseListener 373, 378
mouseDragged method of interface
 MouseMotionListener 373, 381
mouseEntered method of interface
 MouseListener 373
MouseEvent class 353, 373, 628
 `getClickCount` method 380
 `getPoint` method 383
 `getX` method 377
 `getY` method 377
 `isAltDown` method 380
 `isMetaDown` method 380
 `isPopupTrigger` method 628
mouseExited method of interface
 MouseListener 373
MouseInputListener interface 372,
 377
MouseListener interface 353, 372,
 378, 628
 `mouseClicked` method 373, 378
 `mouseEntered` method 373
 `mouseExited` method 373
 `mousePressed` method 373, 628
 `mouseReleased` method 373, 628
MouseMotionAdapter class 378, 381
MouseMotionListener interface 353,
 372, 377, 378
 `mouseDragged` method 373, 381
 `mouseMoved` method 373, 381
mouseMoved method of interface
 MouseMotionListener 373, 381
mousePressed method of interface
 MouseListener 373, 628
mouseReleased method of interface
 MouseListener 373, 628
MouseWheelEvent class 374
MouseWheelListener interface 374
 `mouseWheelMoved` method 374
mouseWheelMoved method of interface
 MouseWheelListener 374
`moveTo` method of class *GeneralPath* 434
 Mozilla Foundation 7
 MP3 player 8
 multi-button mouse 380
 multibyte character set (MBCS) 940
 multi-catch 312
 multi-core 548
 multidimensional array 173, 174
 multiple class declarations
 in one source-code file 200
 multiple document interface (MDI) 612,
 633
 multiple inheritance 235
 multiple-selection list 367, 369, 370
 multiple-selection statement 62
 MULTIPLE_INTERVAL_SELECTION
 constant of interface
 ListSelectionModel 369, 372
 multiplication compound assignment
 operator, `*=` 81
 multiplication, `* 33, 33`
 multiplicative operators: `*, /` and `%` 78
 multiplicity 826, 827
`multiply` method of class *BigDecimal* 231
 multithreading 511, 655, 1005
 multitouch screen 8
 mutable data 664
 mutable reduction (Java SE 8) 563
 mutable reduction operations 553
 mutator method 209
 mutual exclusion 664
 mutually exclusive options 360
 MVC (Model-View-Controller) 801,
 1008
 MySQL 17, 733
- ## N
- `n` conversion character 952
`%n` format specifier (line separator) 29
 name collision 224
 name conflict 224
 name of an array 147
 named constant 151
 naming convention
 GUI (Graphical User Interface)
 component 802
 methods that return `boolean` 105
 native keyword 898
 natural comparison method 520
 natural logarithm 120
 natural order 566
 navigability arrow in the UML 857
`negate` method of functional interface
 Predicate (Java SE 8) 563
 negative arc angles 423
 negative degree 422
 nested array initializers 173
 nested class 194, 347, 629
 relationship between an inner class
 and its top-level class 360
 Nested Class Summary section in API 919
 nested control statement 988
 nested control statements 79, 132
 Examination-results problem 79
 nested for statement 154, 174, 175, 176,
 180, 987
 enhanced for 175
 nested if selection statement 67
 nested if...else selection statement 64,
 65, 67, 69
 nested message in the UML 850
 NetBeans
 Hierarchy window 794, 795
 Inspector window 795
 JavaFX FXML Application project 791
 Library window 795, 796
 Projects window 793
 Netbeans
 demonstration video
 (www.deitel.com/books/jhtp9) 21
 NetBeans (www.netbeans.org) 9
 network message arrival 314
 new keyword 31, 44, 147, 148, 898
 New Project dialog (NetBeans) 792, 802
 new `Scanner(System.in)` expression
 31
 new state 657
 new `CachedThreadPool` method of class
 Executors 661
 new `Condition` method of interface
 Lock 699, 700

newDirectoryStream method of class *Files* 478
newFactory method of interface *RowSetProvider* 767
newline character 27
newline escape sequence, \n 27, 28
newOutputStream method of class *Files* 495, 498
next method

- of *Iterator* 514
- of *ResultSet* 753
- of *Scanner* 44

nextDouble method of class *Scanner* 57
nextInt method of class *Random* 129, 132
nextLine method of class *Scanner* 44
Nimbus look and feel 335

- swing.properties* xl, 335

Nimbus look-and-feel 629
no-argument constructor 205, 206
node 1000, 1001
Node class (JavaFX) 790, 811
node in a JavaFX application 790
non-deterministic random numbers 128
NONE constant of class *GridBagConstraints* 644
nonfatal runtime error 12
NORTH constant of class *BorderLayout* 377, 391
NORTH constant of class *GridBagConstraints* 643
NORTHEAST constant of class *GridBagConstraints* 643
NORTHWEST constant of class *GridBagConstraints* 643
NoSuchElementException class 484, 487
note in the UML 61
Notepad 9
notify method of class *Object* 684
notify method of *Object* 262
notifyAll method of class *Object* 684, 687
notifyAll method of *Object* 262
noun phrase in requirements document 824
now method of class *Instant* 722
null 898
null keyword 45, 50, 147, 337
null reserved word 85
NullPointerException exception 163
Number class 606

- doubleValue** method 607

number systems 461
NumberFormat class 230, 722, 800, 808

- format** method 231, 722
- getCurrencyInstance** method 231
- getPercentInstance** method 722, 809
- setRoundingMode** method 812

numeric Classes 510

O

object 2, 4
object (or instance) 848

Object class 216, 235, 239, 500

- clone** method 262
- equals** method 261
- finalize** method 262
- getClass** method 262, 287, 343
- hashCode** method 262
- notify** method 262, 684
- notifyAll** method 262, 684, 687
- toString** method 242, 262
- wait** method 262, 684

object diagram in the UML 990
object of a derived class 268
object of a derived class is instantiated 261
object-oriented analysis and design (OOAD) 6
object-oriented design (OOD) 816, 822, 824, 834, 857
object-oriented language 6
object-oriented programming (OOP) 2, 6, 233, 235
object serialization 492
ObjectInput interface 492

- readObject** method 493

ObjectInputStream class 492, 493, 498
ObjectOutput interface 492

- writeObject** method 493

ObjectOutputStream class 492, 492, 493, 543, 1006

- close** method 497

Observable class 1003
ObservableValue interface 809
ObservableValue interface (JavaFX)

- addListener** method 812

Observer design pattern 994, 997, 1003
Observer interface 1003
observer object 1003
octal integer 946
octal number system (base 8) 964
of method of interface *IntStream* (Java SE 8) 556
off-by-one error 89
offer method of *PriorityQueue* 533
ON clause 744
ONE constant of class *BigDecimal* 231
one-to-many relationship 738, 738
one-to-one mapping 537
one-, two- or three-button mouse 380
one's complement 971, 980
ones position 964
one-to-many relationship in the UML 829
one-to-one relationship in the UML 829
OOAD (object-oriented analysis and design) 6
OOD (object-oriented design) 816, 822, 824, 834
OOP (object-oriented programming) 6, 233, 235
opaque Swing GUI components 381
open a fil 476
OPEN constant of class *Arc2D* 432
Open Handset Alliance 7
open source 7
operand 77
operating system 7
operation compartment in a class diagram 839
operation in the UML 47, 826, 839, 843, 859, 860, 865, 866
operation parameter in the UML 47, 840, 843, 844, 845
operator 32
operator precedence 33

- operator precedence chart 78
- Operator Precedence Chart
- Appendix 895

rules 33
Operators

- &**, boolean logical exclusive OR 110, 112
- , predecrement/postdecrement 82
- , prefix decrement/postfix
 - decrement 82
- !**, logical NOT 110, 112
- ?:**, ternary conditional operator 66
- *=**, multiplication assignment operator
- /=**, division assignment operator 81
- &**, boolean logical AND 110, 112
- &&**, conditional AND 110, 111
- %=**, remainder assignment operator 81
- ++**, prefix increment/postfix
 - increment 82
- ++**, preincrement/postincrement 81
- +_a**, addition assignment operator 81
- = 32, 36**
- _a**, subtraction assignment operator 81
- |**, boolean logical inclusive OR 110, 112
- ||**, conditional OR 110, 111

arithmetic 33
binary 32, 33
boolean logical AND, & 110, 112
boolean logical exclusive OR, ^ 110, 112
boolean logical inclusive OR, | 112
cast 77
compound assignment 81, 83
conditional AND, && 110, 111, 112
conditional operator, ?: 66
conditional OR, || 110, 111, 112
decrement operator, -- 82
increment and decrement 82
increment, ++ 82
logical complement, ! 112
logical negation, ! 112
logical operators 110, 112, 113

- multiplication, * 33
- multiplicative, *, / and % 78

postfix decrement 82
postfix increment 82
prefix decrement 82
prefix increment 82
remainder, % 33, 33
subtraction, - 33
Optional class (Java SE 8) 570
optional package 227
OptionalDouble class (Java SE 8) 557, 575

- getAsDouble** method 557, 575
- orElse** method 557, 575

or method of class *BitSet* 984
or method of functional interface *Predicate* (Java SE 8) 563

Oracle Corporation 733, 938
ORDER BY SQL clause 739, 742, 743
 ordering of records 739
orElse method of class
OptionalDouble (Java SE 8) 557, 575
 origin component 628
 originator object 998
 out-of-bounds array index 314
 outer set of brackets 157
 output 24
 output cursor 24, 26
 output parameter for a
CallableStatement 785
OutputStream class 493, 503
OutputStreamWriter class 506
 oval 418, 422
 oval bounded by a rectangle 422
 oval filled with gradually changing colors 431
 overflow 314
 overload a method 140
 overloaded constructors 202
 overloaded method 585
 overloading generic methods 593
 override a superclass method 238, 242

P

PaaS (Platform as a Service) 17
pack method of class **Window** 636
 package 30, 118, 127, 222, 990
 package access 228
 package-access members of a class 229
 package-access methods 228
 package declaration 222
 package diagram in the UML 990
 package directory names 224
 package directory structure 222
 package keyword 898
 package-list generated by javadoc 935
 package name 48
 package overview 128
 Packages
 default package 48
 java.time 199
 java.awt 338, 406, 428, 616, 628
 java.awt.color 428
 java.awt.event 127, 350, 352, 377, 387
 java.awt.font 428
 java.awt.geom 127, 428
 java.awt.image 428
 java.awt.image.renderable 428
 java.awt.print 428
 java.beans 720
 java.io 127, 476
 java.lang 31, 119, 127, 239, 261, 437, 659
 java.math 78, 230, 808
 java.nio.file 475, 476, 477, 577
 java.security 128
 java.sql 127, 750, 753
 java.text 230, 800, 808
 java.time 128
 java.util 30, 127, 188

Packages (cont.)
 java.util.concurrent 127, 660, 680, 705, 726
 java.util.concurrent.locks 698, 699
 java.util.function (Java SE 8) 550, 556
 java.util.prefs 541
 java.util.regex 437
 java.util.stream (Java SE 8) 554
 javafx.application.Application 799
 javafx.beans.value 808, 812
 javafx.event 809
 javafx.fxml 809
 javafx.scene 790, 802, 807
 javafx.scene.control 800, 809
 javafx.scene.layout 795, 799
 javafx.stage 790
 javax.sql.rowset 767
 javax.swing 127, 333, 335, 343, 354, 399, 410, 616, 629, 635
 javax.swing.event 127, 351, 352, 369, 377, 616
 javax.swing.table 755, 766
 packages
 create your own 222
 naming 222
 padding (JavaFX) 805
 Padding property of a JavaFX layout container 805
Page Down key 384
 page layout software 437
Page Up key 384
 Paint object 431
 paintComponent method of class **JComponent** 381, 403, 613, 615
 panel 397
 parallel 654
 parallel operations 654
 parallel stream 725
 parallelPrefix method of class Arrays 723
 parallelSetAll method of class Arrays 723
 parallelSort method of class Arrays 188, 721
 parallelSort method of class Arrays (Java SE 8) 563
@param javadoc tag 930
 parameter 45
 parameter in the UML 47, 840, 843, 844, 845
 parameter list 42
 parameter list in a lambda 551
 parameterized class 594
 parameterized type 594
Parameters: note 930
 Parent class (JavaFX) 802, 807
 parent window 337, 612, 633
 parent window for a dialog box 623
 parent window specified as null 623
 parentheses 23
 parseInt method of class **Integer** 337
 parseInt method of **Integer** 184
 pass an array element to a method 165
 pass an array to a method 165
 pass-by-reference 166
 pass-by-value 165, 166
 passing options to a program 184
 password 344
 PATH environment variable xxxviii, 25
 Path interface 477
 getFileName method 478
 isAbsolute method 478
 toAbsolutePath method 478
 toString method 478
 Paths class 477
 get method 477, 478
 pattern 428
 Pattern class 437, 471
 compile method 471
 matcher method 471
 matches method 471
 splitAsStream method (Java SE 8) 578
 pattern matching 740
 Payable interface declaration 293
 Payable interface hierarchy UML class diagram 292
 Payable interface test program
 processing *Invoices* and *Employees* polymorphically 299
 peek method of class **PriorityQueue** 533
 peek method of class **Stack** 533
 percent (%) SQL wildcard character 740
 perform a calculation 37
 perform a task 42
 Performance Tips overview xxxi
 performing operations concurrently 654
 persistent data 475
 persistent **Hashtable** 541
 PHP 17
 physical input operation 505
 physical output operation 504
 PIE constant of class **Arc2D** 432
 pie-shaped arc 432
 pipe 503
 PipedInputStream class 503
 PipedOutputStream class 503
 PipedReader class 506
 PipedWriter class 506
 pixel ("picture element") 403
 PLAF (pluggable look-and-feel) 612
 PLAIN constant of class **Font** 414
 PLAIN_MESSAGE 337
 Platform as a Service (PaaS) 17
 platform dependency 659
 platform independent 10
 pluggable look-and-feel (PLAF) 612
 pluggable look-and-feel package 339
 plus sign (+) indicating public visibility in the UML 857
 PNG (Portable Network Graphics) 343
 point 414
 Point class 382
 poll method of **PriorityQueue** 533
 polygon 425, 427
 Polygon class 403, 425
 addPoint method 426, 428
 polyline 425
 polylines 425
 polymorphic processing
 of collections 511
 polymorphic processing of related exceptions 317

polymorphically process `Invoices` and `Employees` 299

polymorphism 104, 263, 265, 862, 863, 873

pop method of `Stack` 533

popup trigger event 625, 628

portability 405, 940

Portability Tips overview xxii

portable 10

portable GUI 127

Portable Network Graphics (PNG) 343

position number 146

positional notation 964

positional value 965

positional values in the decimal number system 965

positive and negative arc angles 423

positive degrees 422

postcondition 328

postdecrement 82

postfix decrement operator 82

postfix increment operator 82, 91

PostgreSQL 733

postincrement 82, 84

pow method of class `BigDecimal` 231

pow method of class `Math` 95, 96, 119, 120

power (exponent) 120

power of 2 larger than 100 69

prebuilt data structures 508

precedence 33, 37, 84

- arithmetic operators** 34
- chart** 33, 78

Precedence Chart Appendix 895

precise floating-point calculations 230

precision 946, 947

- format of a floating-point number** 78
- precision of a formatted floating-point number** 57

precondition 328

predecrement 82

predefined character class 464

predicate 559, 740

Predicate functional interface (Java SE 8) 551, 569

- and** method 563
- negate** method 563
- or** method 563

predicate method 105, 210

preemptive scheduling 659

Pref Height property of a JavaFX component 796

Pref Width property of a JavaFX component 796

Pref Width property of a JavaFX control 804

Preferences API 541

preferred size (JavaFX) 796

prefix decrement operator 82

prefix increment operator 82

preincrement 82, 84

Preincrementing and postincrementing 82

PreparedStatement interface 769, 770, 772, 776, 785

- executeQuery** method 776
- executeUpdate** method 776
- setString** method 769, 776

prepareStatement method of interface `Connection` 776

previous method of `ListIterator` 517

primary key 734, 738

- composite** 738

prime number 580

primitive type 31, 49, 84, 126

- boolean** 906
- byte** 98
- char** 31, 98
- double** 31, 53, 75
- float** 31, 53
- int** 31, 75, 81, 98
- names are keywords** 31
- passed by value** 167
- promotions** 126
- short** 98

principal in an interest calculation 94

Principle of Least Privilege 140

principle of least privilege 221

print a line of text 24

print debugger command 905

print method of `System.out` 26

print on multiple lines 26

print spooling 673

printArray generic method 588

printf method of `System.out` 28, 945

println method of `System.out` 26

printStackTrace method of class `Throwable` 324

PrintStream class 504, 543

PrintWriter class 484, 506

priority of a thread 658

PriorityBlockingQueue class 706

PriorityQueue class 533

- clear** method 533
- offer** method 533
- peek** method 533
- poll** method 533
- size** method 533

private

- access modifier** 41, 199, 238
- data** 209
- field** 208
- keyword** 209, 857, 898

private static

- class member** 217

probability 129

producer 655, 672

producer thread 673

producer/consumer relationship 672, 692

program construction principles 115

program in the general 265

program in the specific 265

project (NetBeans) 792

Projects window in NetBeans 793

promotion 77

- of arguments** 125
- rules** 78, 125

promotions for primitive types 126

Properties class 541

- getProperty** method 541
- keySet** method 544
- list** method 543
- load** method 544
- setProperty** method 541
- store** method 543

propertyChange method of interface `PropertyChangeListener` 720

PropertyChangeListener interface 720

- propertyChange** method 720
- protected** access modifier 199, 238, 898

protocol for communication (jdbc) 752

Prototype design pattern 994, 995, 1010

Proxy design pattern 994, 996, 997

proxy object 997

public

- abstract** method 290
- access modifier** 40, 41, 122, 199, 238
- final static** data 290
- interface** 194
- keyword** 22, 41, 857, 859, 860, 898
- member of a subclass** 238
- method** 195, 199
- method encapsulated in an object** 198
- service** 194
- static** class members 217
- static** method 217

push method of class `Stack` 532

put method

- of interface** `BlockingQueue` 680, 681
- of interface** `Map` 540

Q

qualified name 744

quantifiers used in regular expressions 468, 469

quantum 658

query 733, 735

query a database 750

query method 209

QUESTION_MESSAGE 337

queue 533

Queue interface 509, 533, 680

R

RadialGradientPaint class 431

radians 120

radio button 354, 360

radio button group 360

radix (base) of a number 461

raised rectangle 422

Random class

- nextInt** method 129, 132

random numbers

- difference between values** 133
- element of chance** 128
- generation** 158
- scaling** 129
- scaling factor** 129, 133
- shift a range** 130
- shifting value** 130, 133

range checking 74

range method of class `EnumSet` 215

range method of interface `IntStream` (Java SE 8) 561

range-view methods 517, 535

rangeClosed method of interface `IntStream` (Java SE 8) 561

Rational Software Corporation 822

Rational Unified Process™ 822
 raw type 602
 read-only file 497
 read-only text 340
 Read/Write Lock design pattern 994, 1005
Reader class 505
 reading files 477
readObject method of **ObjectInput** 493
readObject method of **ObjectInputStream** 500
ready state 658
 real number 31, 75
 realization in the UML 292
 reclaim memory 220
 record 481
 rectangle 403, 407, 419
Rectangle2D class 403
Rectangle2D.Double class 428
 redirect a standard stream 476
reduce method of interface **DoubleStream** (Java SE 8) 575
reduce method of interface **IntStream** (Java SE 8) 557
 reduction (mutable) 563
 reduction operations 553
 reentrant lock 688
ReentrantLock class 698, 700
 refactoring 16
 refer to an object 49
 reference 49
 reference type 49, 228
regexFilter method of class **RowFilter** 766
regionMatches method of class **String** 441
 register an **ActionListener** 624
 register event handlers (JavaFX) 807
 registered listener 353
 registering the event handler 347, 801
 regular expression 464, 576
 ^ 468
 ? 468
 . 472
{*n*, } 469
{*n*, *m*} 468
{*n*} 468
* 468
\d 465
\d 465
\S 465
\s 465
\W 465
\w 465
+ 468
| 468
reinventing the wheel 5, 30, 186
relational database 733, 734
relational database management system (RDBMS) 733
relational database table 734
relational operators 34
relationship between an inner class and its top-level class 360
RELATIVE constant of class **GridBagConstraints** 649
relative path 477
release a lock 664, 686, 687
release a resource 318
release candidate 18
reluctant quantifier 469
remainder 33
remainder compound assignment operator, %≡ 81
REMAINDER constant of class **GridBagConstraints** 649
remainder operator, %≡ 33, 33
remove duplicate **String** 534
remove method of class **ArrayList**<T> 189, 191
remove method of interface **Iterator** 514
removeTableModelListener method of interface **TableModel** 755
Reordering output with argument index 960
repaint method of class **Component** 383
repaint method of class **JComponent** 406
repetition 62
 counter controlled 76
 sentinel controlled 75, 76
 repetition statement 60, 62, 69
 do...while 62, 97, 98, 98
 for 62, 92
 while 62, 69, 70, 72, 75, 76, 88
 repetition terminates 70
replaceAll method
 of class **Matcher** 471
 of class **String** 469
replaceFirst method
 of class **Matcher** 471
 of class **String** 469
representing integers in hexadecimal format 946
representing integers in octal format 946
requestFocus method 811
requestFocus method of class **Node** 811
requirements 6, 820
requirements document 816, 820, 822
requirements gathering 820
requirements of an app 104
reserved word 22, 62, 898
 false 63
 null 45, 50, 85
 true 63
resizable array
 implementation of a **List** 511
resolution 403
resource leak 215, 317
resource-release code 318
responses to a survey 155
result 740
result set concurrency 759
result set type 759
ResultSet interface 753, 759, 760, 761
 absolute method 760
 close method 754
 column name 754
 column number 754
 CONCUR_READ_ONLY constant 759
 CONCUR_UPDATABLE constant 759
 concurrency constant 759
 getInt method 754
 getObject method 754, 760
ResultSet interface (cont.)
 getRow method 761
 last method 761
 next method 753
 TYPE_FORWARD_ONLY constant 759
 TYPE_SCROLL_INSENSITIVE constant 759
 TYPE_SCROLL_SENSITIVE constant 759
ResultSetMetaData interface 753, 760
 getColumnClassName method 760
 getColumnCount method 753, 760
 getColumnName method 760
 getColumnType method 753
ResultSetTableModel enables a **JTable** to display the contents of a **ResultSet** 755
resumption model of exception handling 312
rethrow an exception 321
@return javadoc tag 930
return keyword 42, 125, 898
return message in the UML 850
return type
 in the UML 840, 845
 of a method 41
Returns: note 930
reusability 594
reusable software components 4, 127, 236
reuse 5, 30
reverse method of class **StringBuilder** 454
reverse method of **Collections** 519, 525
reversed method of interface **Comparator** (Java SE 8) 566
reverseOrder method of **Collections** 521
RGB value 406, 407, 412
right aligned 389
right brace, { 23, 30, 72, 76
RIGHT constant of class **FlowLayout** 391
right justification 945, 954
right justify output 95
right justifying integers 954
right-align the contents of a column 804
rigid area of class **Box** 642
robust 32
robust application 305
role in the UML 827
role name in the UML 827
roll back a transaction 786
rollback method of interface **Connection** 786
rolling two dice 136
rollover **Icon** 356
root directory 477
root node 790
rotate method of class **Graphics2D** 435
round a floating-point number for display purposes 78
round-robin scheduling 658
rounded rectangle 420, 432
rounded rectangle (for representing a state in a UML state diagram) 835
rounding 945

rounding a number 33, 73, 96, 120
RoundingMode enum 232, 808
RoundRectangle2D class 403
RoundRectangle2D.Double class 428, 432
 row 734, 738, 739, 740, 741, 745
RowFilter class 766
 rows of a two-dimensional array 173
 rows to be retrieved 739
RowSet interface 767
RowSetFactory class 767
RowSetFactory interface
 createJdbcRowSet method 767
RowSetProvider class 767
RowSetProvider interface
 newFactory method 767
 Rule of Entity Integrity 738
 Rule of Referential Integrity 737
 rule of thumb (heuristic) 109
 rules of operator precedence 33
 run debugger command 903
 run method of interface **Runnable** 659
 runAsync method of class
 CompletableFuture 730
Runnable interface 301, 659
 run method 659
 runnable state 657
 running an application 13
 running state 658
 runtime error 12
 runtime logic error 32
RuntimeException class 315

S

SaaS (Software as a Service) 17
SalariedEmployee class that
 implements interface **Payable**
 method **getPaymentAmount** 297
SalariedEmployee concrete class
 extends abstract class **Employee** 277
 SAM interface 550
 SansSerif Java font 414
 saturation 412
 savings account 94
 scalar 165
 scaling (random numbers) 129
 scale factor 129, 133
 scaling **BigDecimal** values 232
Scanner class 30, 31
 hasNext method 101
 next method 44
 nextDouble method 57
 nextLine method 44
 Scene Builder 789, 791
Scene class (JavaFX) 790, 799, 807, 808
 scene graph 808
 scene graph in a JavaFX application 790
 scene in a JavaFX application 790
 scheduling threads 658
 scientific notation 947
 scope 90
 scope of a declaration 138
 scope of a type parameter 596
 scope of a variable 90
Screen class (ATM case study) 826, 828, 839, 846, 847, 848, 849, 850, 851, 859
 screen cursor 28
 screen-manager program 267
 script (Unicode) 943
 scroll 365, 369
 scroll arrow 366
 scroll box 366
SCROLL_TAB_LAYOUT constant of class
 JTabbedPane 642
 scrollbar 369, 401
 of a **JComboBox** 366
 scrollbar policies 401
 SDK (Software Development Kit) 17
 secondary storage devices 475
 sector 423
SecureRandom class 128, 129
 documentation 129
 doubles method (Java SE 8) 580
 ints method (Java SE 8) 580
 longs method (Java SE 8) 580
 streams of random numbers (Java SE 8) 580
 security 11
 security breach 45
 security breaches 129
SecurityException class 483
See Also: note 929
@see javadoc tag 929
seful 477
SELECT SQL keyword 739, 740, 741, 742, 743
selectAll method 811
selectAll method of class
 TextInputControl 811
 selected text in a **JTextArea** 400
 selecting an item from a menu 344, 801
 selection 62
 selection criteria 740
 selection mode 369
 selection statement 60, 61
 if 61, 62, 63, 98
 if...else 61, 63, 63, 64, 75, 98
 switch 61, 98, 103
 semicolon (;) 24, 31, 36
 send a message to an object 5
 sentence-style capitalization 336
 sentinel-controlled repetition 74, 75, 76
 sentinel value 74, 76
 separator character 480
 separator line in a menu 623, 624
 sequence 62, 511
 sequence diagram in the UML 823, 848
 sequence of messages in the UML 849
 sequence structure 60
 sequence-structure activity diagram 61
SequenceInputStream class 505
 sequential-access file 475, 481
Serializable interface 301, 493
 serialized object 492
 serialized-form.html generated by
 javadoc 935
Serif Java font 414
 service of a class 199
 set debugger command 905
Set interface 509, 534, 535, 537
 stream method (Java SE 8) 578
 set method
 of class **BitSet** 983
 of interface **ListIterator** 517
 set method 202

SET SQL clause 746
 set up event handling 347
setAlignment method of class
 FlowLayout 391
setAutoCommit method of interface
 Connection 786
setBackground method of class
 Component 370, 412
setBounds method of class **Component** 388
setCharAt method of class
 StringBuilder 454
setColor method of class **Graphics** 407, 432
setCommand method of **JdbcRowSet**
 interface 769
setConstraints method of class
 GridBagLayout 649
setDefaultCloseOperation method
 of class **JFrame** 344, 616
setDisabledTextColor method of
 class **JTextComponent** 387
setEditable method of class
 JTextComponent 347
setErr method of class **System** 476
setFileSelectionMode method of
 class **JFileChooser** 500
setFixedCellHeight method of class
 JList 372
setFixedCellWidth method of class
 JList 372
setFont method of class **Component** 359
setFont method of class **Graphics** 414
setForeground method of class
 JComponent 624
setHorizontalAlignment method of
 class **JLabel** 343
setHorizontalScrollBarPolicy
 method of class **JScrollPane** 401
setHorizontalTextPosition
 method of class **JLabel** 343
setIcon method of class **JLabel** 343
setIn method of class **System** 476
setInverted method of class **JSlider** 613
setJMenuBar method of class **JFrame** 617, 624
setLayout method of class **Container** 342, 389, 393, 396, 642
setLength method of class
 StringBuilder 453
setLineWrap method of class
 JTextArea 401
setListData method of class **JList** 372
 setLocation method of class
 Component 388, 617
setLookAndFeel method of class
 UIManager 632
setMajorTickSpacing method of class
 JSlider 616
setMaximumRowCount method of class
 JComboBox 366
setMnemonic method of class
 AbstractButton 623
setOpaque method of class
 JComponent 381, 383
setOut method of **System** 476

setPaint method of class `Graphics2D` 431
setPaintTicks method of class `JSlider` 616
setPassword method of `JdbcRowSet` interface 769
setProperty method of `Properties` 541
setRolloverIcon method of class `AbstractButton` 357
setRoundingMode method of class `NumberFormat` 812
setRowFilter method of class `JTable` 766
setRowSorter method of class `JTable` 766
setScale method of class `BigDecimal` 232
setSelected method of class `AbstractButton` 624
setSelectionMode method of class `JList` 369
setSize method of class `Component` 388, 617
setSize method of class `JFrame` 344
setString method of interface `PreparedStatement` 769, 776
setStroke method of class `Graphics2D` 431
setText method 811
setText method of class `JLabel` 343
setText method of class `JTextComponent` 400
setText method of class `TextInputControl` 811
Setting the PATH environment variable xxxviii
setToolTipText method of class `JComponent` 342
setUrl method of `JdbcRowSet` interface 767
setUsername method of `JdbcRowSet` interface 769
setVerticalAlignment method of class `JLabel` 343
setVerticalScrollBarPolicy method of class `JScrollPane` 401
setVerticalTextPosition method of class `JLabel` 343
setVisible method of class `Component` 344, 393, 617
setVisibleRowCount method of class `JList` 369
shadow 42
shadow a field 138
shallow copy 262
shape 428
Shape class hierarchy 237
Shape object 431
share memory 655
shared buffer 673
shell 24
shell in Linux 10
shell script 483
Shift 387
shift (random numbers) 130
 shifting value 130, 133
short-circuit evaluation 111
Short class 510
short primitive type 98, 898, 899
 promotions 126
short-circuiting terminal operation (Java SE 8) 570
shortcut key 618
show method of class `JPopupMenu` 628
showDialog method of class `JColorChooser` 411
showInputDialog method of class `JOptionPane` 336
showMessageDialog method of class `JOptionPane` 337
showOpenDialog method of class `JFileChooser` 500
shuffle 158
 algorithm 523
shuffle method of class `Collections` 519, 523, 525
shutdown method of class `ExecutorService` 663
side effect 112
Sieve of Eratosthenes 580, 716
 Sieve of Eratosthenes, using a `BitSet` 984
signal method of interface `Condition` 699, 703
signal value 74
signalla11 method of interface `Condition` 699
signature of a method 142, 142
signed right shift (>>) 973, 974, 980, 983
simple condition 110
simple name 224
simulate a middle-mouse-button click on a one- or two-button mouse 380
simulate a right-mouse-button click on a one-button mouse 380
simulation 128
sin method of class `Math` 120
@since javadoc tag 932
Since: note 932
sine 120
single abstract method (SAM) interface 550
single-entry/single-exit control statements 62
single inheritance 235
single-precision floating-point number 53
single-quote character 437, 741
single-selection list 367
single-selection statement 61, 62
single static import 220
Single-Threaded Execution design pattern 994, 1005
single-type-import declaration 226
SINGLE_INTERVAL_SELECTION constant of interface `ListSelectionModel` 369, 370, 372
SINGLE_SELECTION constant of interface `ListSelectionModel` 369
Singleton design pattern 994, 995
size method
 of class `ArrayBlockingQueue` 682
 of class `ArrayList<T>` 191
 of class `BitSet` 984
 of class `Files` 478
 of class `PriorityQueue` 533
 of interface `List` 513, 517
 of interface `Map` 541
skinning 789
sleep interval 657
sleep method of class `Thread` 661, 674, 676, 677
sleeping thread 657
Slider class (JavaFX) 798, 800
 Max property 806
 Value property 806
 valueProperty method 812
small circles in the UML 61
small diamond symbol (for representing a decision in a UML activity diagram) 838
smartphone 3
snap-to ticks for `JSlider` 612
Socket class 1006
SocketImpl class 1006
Software as a Service (SaaS) 17
Software Development Kit (SDK) 17
software engineering 209
Software Engineering Observations overview xxii
software life cycle 820
software reuse 5, 222
solid circle (for representing an initial state in a UML diagram) in the UML 835, 836
solid circle enclosed in an open circle (for representing the end of a UML activity diagram) 836
solid circle in the UML 61
solid circle surrounded by a hollow circle in the UML 61
solid diamonds (representing composition) in the UML 827
sort method
 of class `Arrays` 186
 of class `Collections` 520
sort method of class `Arrays` 563, 721
sorted method of interface `InputStream` (Java SE 8) 559
sorted method of interface `Stream` (Java SE 8) 563, 566
sorted order 535, 537
SortedMap interface 537
SortedSet interface 535, 536
 first method 536
 last method 536
sorting
 descending order 520
 with a `Comparator` 521
source code 9
SourceForge 7
SOUTH constant of class `BorderLayout` 377, 391
SOUTH constant of class `GridBagConstraints` 643
SOUTHEAST constant of class `GridBagConstraints` 643
SOUTHWEST constant of class `GridBagConstraints` 643
space character 22
space flag 957
special character 31, 437
specialization 235
specialization in the UML 863
specifics 267
split method of class `String` 463, 469

splitAsStream method of class `Pattern` (Java SE 8) **578**
SQL 733, 735, 739, 745
 DELETE statement 739, **747**
 FROM clause 739
 GROUP BY 739
 IDENTITY keyword 735
 INNER JOIN clause 739, **744**
 INSERT statement 739, **745**
 LIKE clause 741
 ON clause 744
 ORDER BY clause 739, **742**, 743
 SELECT query 739, 740, 741, 742,
 743
 SET clause **746**
 UPDATE statement 739
 VALUES clause 745
 WHERE clause 740
 SQL (Structured Query Language) **769**
 SQL injection attack (preventing) **770**
 SQL keyword 739
 SQL statement **786**
 SQLException class **752**, 754, **770**
SQLFeatureNotSupportedException
 Exception class **760**
 sqrt method of class `Math` 119, 120, 125
 square brackets, `[]` **146**
 square root 120
 stack 594
Stack class 533
 isEmpty method **533**
 of package `java.util` **531**
 peek method **533**
 pop method **533**
 push method **532**
Stack generic class 594
 Stack<`Double`> **601**
 Stack<`Integer`> **601**
Stack generic class declaration 595
 stack trace **307**
 stack unwinding **322**
StackTraceElement class 324
 getClassName method 324
 getFileName method 324
 getLineNumber method 324
 getMethodName method 324
Stage class (JavaFX) **790**, 799, 807, 808
 stage in a JavaFX application **790**
 stale value **670**
 standard error stream **311**, **320**, **945**
 standard error stream (`System.err`)
 476, 504
 standard input stream (`System.in`) **31**,
 476
 standard output stream **320**
 standard output stream (`System.out`)
 24, 476, 504
 standard reusable component **236**
 standard time format 197
start method of class `Application`
 (JavaFX) **799**, 807
 starting angle 422
startsWith method of class `String`
 444
 starvation **659**
 state **823**
 state button 357
 state dependent 673
 State design pattern 994, 997, 998, 1004
 context object 998
 State class 998
 state object 998
 State subclass 998
 state diagram for the ATM object **835**
 state diagram in the UML **835**
 state in the UML **823**, **836**
 state machine diagram in the UML **823**,
 835
 state object 1004
 state of an object **830**, **835**
stateChanged method of interface
`ChangeListener` **616**
 stateful intermediate operation **560**
 stateless intermediate operation **560**
 stateless stream operation **560**
 statement 24, 42
 statement block in a lambda **551**
Statement interface **753**, **754**, **769**
 close method **754**
 executeQuery method **753**
Statements
 break **102**, **108**
 continue **108**, **988**
 control statement 60, 62, 63
 control-statement nesting **62**
 control-statement stacking **62**
 do...while **62**, 97, 98
 double selection **62**
 empty **37**, **66**
 empty statement **66**
 enhanced for **163**
 for **62**, **88**, 91, 92, 94, 95
 if **34**, **61**, **62**, **63**, **98**
 if...else **61**, **63**, **63**, **64**, **75**, **98**
 labeled break **987**
 labeled continue **988**
 looping **62**
 multiple selection **62**
 nested **79**
 nested if...else **64**, **65**
 repetition **60**, **62**, **69**
 return **125**
 selection **60**, **61**
 single selection **61**
 switch **61**, **98**, **103**, **132**
 throw **196**
 try **157**
 try-with-resources **330**
 while **62**, **69**, **70**, **72**, **75**, **76**, **88**
static
 class member 216, 217
 class variable 217
 field (class variable) **216**
 import **220**
 import on demand **220**
 keyword 119, 898
 method 47, 95
 static binding **289**
static interface methods (Java SE 8)
 302
 static method in an interface (Java SE
 8) **550**, **581**
 static method of an interface (Java SE
 8) **548**
 step debugger command **907**
 step up debugger command **908**
 stop debugger command **903**
store method of `Properties` **543**
 stored procedure **785**
 Strategy design pattern 994, 997, 1004
 strategy object 1004
 stream 320, 945
 stream (Java SE 8)
 DoubleStream interface **554**
 eager operations **557**
 filter elements **559**
 intermediate operation **552**
 IntStream interface **554**
 lazy operation **559**, **560**
 LongStream interface **554**
 map elements to new values **560**
 pipeline **552**, **559**, **560**
 terminal operation **552**
Stream interface (Java SE 8) **552**, **562**
 collect method **563**, **563**, **573**,
 574, 580
 distinct method **572**
 filter method **563**, **566**
 findFirst method **570**
 forEach method **563**
 map method **565**, **565**
 sorted method **563**, **566**
Stream interface (Java SE 8)
 flatMap method **578**
stream method of class `Arrays` (Java SE
 8) **561**, **562**
stream method of interface `Set` **578**
 stream of bytes **475**
 stream pipeline **556**
 streaming **655**
 streams **552**
 streams (Java SE 8)
 infinite streams **580**
strictfp keyword **898**
 string 24
 literal 24
 of characters 24
String class 437
 charAt method **439**, **454**
 compareTo method **441**, **443**
 concat method **448**
 endsWith method **444**
 equals method **441**, **443**
 equalsIgnoreCase method **441**,
 443
 format method **196**, **962**
 getChars method **439**
 immutable **219**
 indexOf method **445**
 lastIndexOf method **445**
 length method **439**
 matches method **464**
 regionMatches method **441**
 replaceAll method **469**
 replaceFirst method **469**
 split method **463**, **469**
 startsWith method **444**
 substring method **447**
 toCharArray method **450**
 toLowerCase **517**
 toLowerCase method **450**
 toUpperCase **517**
 toUpperCase method **449**
 trim method **450**
 valueOf method **450**
String class searching methods 445

- string concatenation 123, 219
 string literal 437
StringBuffer class 452
StringBuilder class 437, 451
 - append method 455
 - capacity method 452
 - charAt method 454
 - constructors 452
 - delete method 457
 - deleteCharAt method 457
 - ensureCapacity method 453
 - getChars method 454
 - insert method 457
 - length method 452
 - reverse method 454
 - setCharAt method 454
 - setLength method 453**StringIndexOutOfBoundsException** class 447
StringReader class 506
StringWriter class 506
Stroke object 431, 432
 strongly typed languages 84
 Stroustrup, Bjarne 305
 structured design patterns 993, 996, 999, 1006
 structure of a system 834, 835
 structured programming 60, 87, 109
 Structured Query Language (SQL) 733, 735, 739
 subclass 5, 235, 862, 863, 1000
 subject object 1003
 sublist 517
subList method of *List* 517
 submenu 618
submit method of class *ExecutorService* 726
 subprotocol for communication 752
 subscript (index) 146
substring method of class *String* 447
 subsystem 1007
 subtraction 33
 - operator, - 33
 subtraction compound assignment operator, -= 81
 suffix F for *Float* literals 532
 suffix L for *long* literals 532
sum method of interface *DoubleStream* (Java SE 8) 575
sum method of interface *IntStream* (Java SE 8) 557
 sum the elements of an array 152
 summarizing responses to a survey 155
 Sun Microsystems 938, 993
super keyword 238, 261, 898
 - call superclass constructor 252
 superclass 5, 235, 862, 1001
 - constructor 242
 - constructor call syntax 252
 - default constructor 242
 - direct 235, 237
 - indirect 235, 237
 - method overridden in a subclass 260**Supplier** functional interface (Java SE 8) 551
Supplier interface (Java SE 8) 726, 729
supplyAsync method of class *CompletableFuture* 729
 surrogates 938
- sweep 422
 sweep counterclockwise 422
Swing Event Package 127
Swing GUI APIs 334
Swing GUI components 333
Swing GUI components package 127
swing.properties file xl, 335
SwingConstants interface 343, 616
SwingUtilities class 632
 - updateComponentTreeUI method 632**SwingWorker** class 707
 - cancel method 721
 - doInBackground method 707, 710
 - done method 707, 710
 - execute method 707
 - get method 707
 - isCancelled method 716
 - process method 708, 717
 - publish method 707, 717
 - setProgress method 708, 716
 switch logic 104
 switch multiple-selection statement 61, 98, 103, 132, 898
 - activity diagram with break statements 103
 - case label 102
 - controlling expression 102
 - default case 102, 104, 132**Sybase** 733
Sybase, Inc. 938
 synchronization 664, 684
 synchronization wrapper 544
 synchronize 655
 synchronize access to a collection 511
synchronized
 - keyword 544, 664, 898
 - method 665
 - statement 664
 synchronized collection 511
 synchronous call 848
 synchronous error 314
SynchronousQueue class 706
 system 822
 system behavior 822
System class
 - arraycopy* 186, 187
 - exit* method 318, 483
 - setErr* method 476
 - setIn* method 476
 - setOut* 476
 system requirements 820
 system structure 822
System.err (standard error stream) 311, 476, 504, 945
System.in (standard input stream) 476
System.out
 - print* method 26, 26, 26
 - printf* method 28
 - println* method 24, 26**System.out** (standard output stream) 24, 476, 504
SystemColor class 431
- T**
- tab 960
 tab character, \t 28
Tab key 23
- tab stops 28
 table 173, 734
 table element 173
 table of values 173
TableModel interface 754
 - addTableModelListener* 755
 - getColumnClass* method 755, 760
 - getColumnCount* method 755, 760
 - getColumnName* method 755, 760
 - getRowCount* method 755
 - getValueAt* method 755
 - removeTableModelListener* 755**TableModelEvent** class 766
TableRowSorter class 766
 tabular format 150
 tagging interface 291, 493
tailSet method of class *TreeSet* 536
 take method of class *BlockingQueue* 680, 682
tan method of class *Math* 120
 tangent 120
 target type of a lambda (Java SE 8) 556
 technical publications 18
 Template Method design pattern 994, 997, 1004
 temporary 77
TEN constant of class *BigDecimal* 231
Terminal application (OS X) 10
 terminal operation 556
 - eager 559
 terminal operations
 - mutable reduction 553
 - reduction 553
 terminal stream operations (Java SE 8) 552, 563
 - average method of interface *IntStream* 557
 - collect method of interface *Stream* 563, 573, 574, 580
 - count method of interface *IntStream* 557
 - findFirst method of interface *Stream* 570
 - mapToDouble* method of interface *Stream* 574
 - max method of interface *IntStream* 557
 - min method of interface *IntStream* 557
 - reduce method of interface *IntStream* 557
 - short-circuiting 570
 - sum method of interface *IntStream* 557
 terminal window 24
 terminate an application 623
 terminate successfully 483
terminated state 657
 termination housekeeping 216, 262
 termination model of exception handling 312
 ternary operator 66
test method of functional interface *IntPredicate* (Java SE 8) 559, 560
 text editor 24, 437
 text file 476
Text property of a *Label* (JavaFX) 796
TextEdit 9
TextField class (JavaFX) 800, 804

TextInputControl class (JavaFX) 811
TexturePaint class 403, 431, 432
The Java™ Language Specification (java.sun.com/docs/books/jls/) 33
thenComparing method of functional interface **Comparator** (Java SE 8) 571
thick lines 428
this
 keyword 42, 200, 217, 898
 reference 200
 to call another constructor of the same class 205
thread 312, 405, 1005
 life cycle 656, 658
 of execution 655
 scheduling 658, 677
 state **656**
 synchronization 544, 664
Thread class
 currentThread method 661, 666
 interrupt method 661
 sleep method 661
thread confinement 707
thread-life-cycle statechart diagram 656, 658
thread pool 659
thread priority 658
thread safe 670, 707
thread scheduler 658
thread states
 blocked 657, 665
 dead 657
 new 657
 ready 658
 runnable 657
 running 658
 terminated 657
 timed waiting 657
 waiting 657
three-button mouse 380
three-dimensional rectangle 419
throw an exception 157, 306, 310
throw an exception 196, 206
throw keyword 321, 898
throw point 308
throw statement 320
Throwable class 314, 324
 getMessage method 324
 getStackTrace method 324
 hierarchy 315
 printStackTrace method 324
throws clause 313
@**throws javadoc tag** 930
throws keyword 898
thumb of class **JSlider** 612, 616
thumb position of class **JSlider** 616
tick marks on a **JSlider** 612
time formatting 946
timed waiting state 657
timeslice 658
timeslicing 658
timing diagram in the UML 991
title bar 334, 340, 616
title bar of a window 337
title bar of internal window 636
titles table of books database 735, 736
toAbsolutePath method of interface **Path** 478
toArray method of **List** 518, 519
toBinaryString method of class **Integer** 976
toCharArray method of class **String** 450
ToDoubleFunction functional interface (Java SE 8) 575
 applyAsDouble method 575
toggle button 354
token of a **String** 463
tokenization 463
toList method of class **Collectors** (Java SE 8) 563
toLowerCase method of class **Character** 461
toLowerCase method of class **String** 450, 517
toMillis method of class **Duration** 722
tool tips 339, 342, 344
top 533
TOP constant of class **JTabbedPane** 642
top-level class 347
toPath method of class **File** 501
toString method
 of class **ArrayList** 520, 607
 of class **Arrays** 471
 of class **BitSet** 984
 of class **Formatter** 962
 of class **Object** 242, 262
toString method of an object 124
toString method of interface **Path** 478
toUpperCase method of class **Character** 460
toUpperCase method of class **String** 449, 517
track mouse events 374
traditional comment 22
trailing white-space characters 450
Transaction class (ATM case study) 862, 863, 864, 865, 867, 893
transaction processing 785, 786
transient keyword 495, 898
transition arrow 63, 70
 in the UML 61
transition arrow in the UML 70
transition between states in the UML 835, 838
transition in the UML 61
translate method of class **Graphics2D** 435
transparency of a **JComponent** 381
traverse an array 174
tree 534
Tree link in API 916
TreeMap class 537, 578
TreeSet class 534, 535, 536
 headSet method 535
 tailSet method 536
trigger an event 338
trigonometric cosine 120
trigonometric sine 120
trigonometric tangent 120
trim method of class **String** 450
trimToSize method of class **ArrayList<T>** 189
true 34, 898
true reserved word 63, 66
truncate 33
truncate fractional part of a calculation 73
truncated 481
truth table 110
truth tables
 for operator **^** 112
 for operator **!** 113
 for operator **&&** 110
 for operator **||** 111
try block 157, 310, 322
 terminates 312
try keyword 310, 898
try statement 157, 313
try-with-resources statement 330
24-hour clock format 194
two-dimensional array 173, 174
two-dimensional graphics 428
two-dimensional shapes 403
Two-Phase Termination design pattern 994, 1006
two's complement 971
two position 966
type 31
type argument 596
type casting 77
type-import-on-demand declaration 226
type inference with the **<:** notation (Java SE 7) 514
type inferencing 514
type of a lambda expression 551
type parameter 588, 594, 601
 scope 596
 section 588, 594
type safety 587
type variable 588
type-wrapper class 458, 510, 590
 implements Comparable 590
TYPE_FORWARD_ONLY constant 759
TYPE_INT_RGB constant of class **BufferedImage** 432
TYPE_SCROLL_INSENSITIVE constant 759
TYPE_SCROLL_SENSITIVE constant 759
Types class 754
typesetting system 437
type-wrapper classes 458
typing in a text field 344, 801

U

U+yyyy (Unicode notational convention) 939
UIManager class 629
 getInstalledLookAndFeels method 629
 LookAndFeelInfo nested class 629
 setLookAndFeel method 632
UIManager.LookAndFeelInfo class
 getClassName method 632
UML (Unified Modeling Language) 6, 816, 822, 826, 833, 834, 862
 activity diagram 60, 61, 63, 70, 92, 98
 aggregation 828
 arrow 61
 association 826
 class diagram 46

- UML (cont.)
- compartment in a class diagram 46
 - diagram 822
 - diamond 62
 - dotted line 61
 - elided diagram 826
 - final state 61
 - frame 850
 - guard condition 62
 - hollow diamond representing aggregation 828
 - many-to-one relationship 829
 - merge symbol 70
 - multiplicity 826
 - note 61
 - one-to-many relationship 829
 - one-to-one relationship 829
 - Resource Center (www.deitel.com/UML/) 823
 - role name 827
 - solid circle 61
 - solid circle surrounded by a hollow circle 61
 - solid diamond representing composition 827
 - Specification (www.omg.org/technology/documents/formal/uml.htm) 828
- UML Activity Diagram
- small diamond symbol (for representing a decision) in the UML 838
 - solid circle (for representing an initial state) in the UML 836
 - solid circle enclosed in an open circle (for representing the end of an activity) in the UML 836
- UML Class Diagram 826
- attribute compartment 833
 - operation compartment 839
- UML Sequence Diagram
- activation 850
 - arrowhead 850
 - lifeline 850
- UML State Diagram
- rounded rectangle (for representing a state) in the UML 835
 - solid circle (for representing an initial state) in the UML 835
- UML Use Case Diagram
- actor 821
 - use case 822
- unambiguous (Unicode design basis) 938
- unary operator 78, 112
- cast 77
- UnaryOperator** functional interface (Java SE 8) 551
- unbiased shuffling algorithm 161
- unboxing 594, 599
- unboxing conversion 510
- uncaught exception 312
- unchecked exceptions 315
- uncovering a component 405
- underlying data structure 533
- underscore (_) SQL wildcard character 740, 741
- uneditable **JTextArea** 398
- unreadable text or icons 338
- Unicode character set 85, 104, 437, 442, 460, 899
- Unicode Consortium 938
- Unicode Standard 937
- Unicode Standard design basis 938
- Unicode value of the character typed 387
- Unified Modeling Language (UML) 6, 816, 822, 826, 833, 834, 862
- uniform (Unicode design principle) 938
- Uniform Resource Identifier (URI) 478
- Uniform Resource Locator (URL) 478
- universal (Unicode design principle) 938
- universal-time format 194, 196, 197
- UNIX 24, 101, 483
- unlock method of interface **Lock** 698, 703
- unmodifiable collection 511
- unmodifiable wrapper 544
- unsigned right shift (>>>) 973, 974, 980, 983
- unspecified number of arguments 182
- UnsupportedOperationException** class 517
- unwatch debugger command 911
- unwinding the method-call stack 322
- UPDATE SQL statement 739, 746
- updateComponentTreeUI method of class **SwingUtilities** 632
- upper bound 591
- of a wildcard 607
- upper bound of a type parameter 592, 593
- upper-left corner of a GUI component 403
- upper-left x-coordinate 407
- upper-left y-coordinate 407
- uppercase letter 23, 31
- URI (Uniform Resource Identifier) 478
- URL 1007
- URL (Uniform Resource Locator) 478
- URLStreamHandler** class 1007
- use case diagram in the UML 821, 822
- use case in the UML 821
- use case modeling 821
- user interface 1009
- UTF-16 938
- UTF-32 938
- UTF-8 938
- Utilities Package 127
- Utility class that displays bit representation of an integer 979
- utility method 197
- V**
- va 484
- vacated bits 983
- valid identifier 31
- validate data 74
- validate method of class **Container** 396
- validity checking 209
- Value** property of a **Slider** (JavaFX) 806
- valueChanged method of interface **ListSelectionListener** 369
- valueOf method of class **BigDecimal** 230
- valueOf method of class **String** 450
- valueProperty method of class **Slider** 812
- values method of an enum 214
- VALUES SQL clause 745, 745
- variable 31
- name 31
 - reference type 49
- variable declaration statement 31
- variable is not modifiable 221
- variable-length argument list 182
- variable names
- came case naming 40
- variable scope 90
- VBox** class (JavaFX) 795
- alignment 796
 - Alignment property 796
- Vbox** class (JavaFX) 795
- Vector** class 511
- verb phrase in requirements document 839
- @version javadoc tag 932
- Version note 932
- VERTICAL** constant of class **GridBagConstraints** 644
- vertical coordinate 403
- vertical gap space 393
- vertical scrolling 400
- vertical strut 641
- VERTICAL_SCROLLBAR_ALWAYS**
- constant of class **JScrollPane** 401
- VERTICAL_SCROLLBAR_AS_NEEDED**
- constant of class **JScrollPane** 401
- VERTICAL_SCROLLBAR_NEVER** constant of class **JScrollPane** 401
- Vgap** property of a **GridPane** 805
- vi editor 9
- video game 129
- video streaming 691
- View** 334
- view (in MVC) 801, 1008
- virtual key code 387
- virtual machine (VM) 10
- visibility in the UML 857
- visibility marker in the UML 857
- Visitor design pattern 997
- visual feedback 357
- visual programming 791
- Vlissides, John 993
- void keyword 23, 41, 898
- volatile keyword 898
- W**
- wait method of class **Object** 262, 684
- waiting line 533
- waiting state 657
- waiting thread 687
- watch debugger command 909
- waterfall model 820
- weights** field of class **GridBagConstraints** 644
- weighty** field of class **GridBagConstraints** 644
- WEST** constant of class **BorderLayout** 377, 391
- WEST** constant of class **GridBagConstraints** 643
- WHERE SQL clause 739, 740, 741, 743, 746, 747

w
while repetition statement 62, 69, 70, 72, 75, 76, 88
 activity diagram in the UML 70
white space 22, 24
white-space character 450, 463, 464
whole/part relationships 827
widgets 333, 788
width 418
width of a rectangle in pixels 407
wildcard 607
 in a generic type parameter 605
 type argument 607, 607
 upper bound 607
window 616
Window class 616, 617
 addWindowListener method 617
 dispose method 617
 pack method 636
window event 617
window event-handling methods 377
window events 617
window gadgets 333, 788
windowActivated method of interface
 WindowListener 617
WindowAdapter class 378, 766
windowClosed method of interface
 WindowListener 617, 766
windowClosing method of interface
 WindowListener 617
WindowConstants interface 616
 DISPOSE_ON_CLOSE constant 617
 DO NOTHING_ON_CLOSE constant 616
 HIDE_ON_CLOSE constant 616
windowDeactivated method of
 interface **WindowListener** 617
windowDeiconified method of
 interface **WindowListener** 617
windowIconified method of interface
 WindowListener 617
windowing system 339
WindowListener interface 377, 378, 617, 766
 windowActivated method 617
 windowClosed method 617, 766
 windowClosing method 617
 windowDeactivated method 617
 windowDeiconified method 617
 windowIconified method 617
 windowOpened method 617
windowOpened method of interface
 WindowListener 617
Windows 101, 483
Windows look-and-feel 612
Withdraw class (ATM case study)
 826, 827, 829, 831, 832, 837, 838,
 839, 847, 848, 850, 851, 859, 860,
 862, 863, 864, 867
word character 464
word processor 437, 445
workflow 60
workflow of an object in the UML 836
wrapper methods of the **Collections**
 class 511
wrapper object (collections) 544
wrapping stream objects 492, 497
wrapping text in a JTextArea 401
writeBoolean method of interface
 DataOutput 504
writeByte method of interface
 DataOutput 504
writeBytes method of interface
 DataOutput 504
writeChar method of interface
 DataOutput 504
writeChars method
 of interface **DataOutput** 504
writeDouble method
 of interface **DataOutput** 504
writeFloat method
 of interface **DataOutput** 504
writeInt method of interface
 DataOutput 504
writeLong method of interface
 DataOutput 504
writeObject method
 of class **ObjectOutputStream** 497
 of interface **ObjectOutput** 493
Writer class 505i
writeShort method of interface
 DataOutput 504
writeUTF method of interface
 DataOutput 504
www 16

X
x-coordinate 403, 427
X_AXIS constant of class **Box** 642
x-axis 403
xor method of class **BitSet** 984

Y
y-coordinate 403, 427
Y_AXIS constant of class **Box** 642
y-axis 403

Z
ZERO constant of class **BigDecimal** 231
0 (zero) flag 956, 958
zero-based counting 149
zeroth element 146

This page intentionally left blank