Java SE 8 for Programmers

Paul Deitel • Harvey Deitel

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

facebook twitter googleplus linkedin stumbleupon
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

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
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.
Foreword

Preface

Before You Begin

I Introduction to Java and Test-Driving a Java Application

1.1 Introduction

1.2 Object Technology Concepts

1.2.1 The Automobile as an Object

1.2.2 Methods and Classes

1.2.3 Instantiation

1.2.4 Reuse

1.2.5 Messages and Method Calls

1.2.6 Attributes and Instance Variables

1.2.7 Encapsulation and Information Hiding

1.2.8 Inheritance

1.2.9 Interfaces

1.2.10 Object-Oriented Analysis and Design (OOAD)

1.2.11 The UML (Unified Modeling Language)

1.3 Open Source Software

1.4 Java

1.5 A Typical Java Development Environment

1.6 Test-Driving a Java Application

1.7 Software Technologies

1.8 Keeping Up-to-Date with Information Technologies

II Introduction to Java Applications; Input/Output and Operators

2.1 Introduction

2.2 Your First Program in Java: Printing a Line of Text

2.3 Modifying Your First Java Program

2.4 Displaying Text with printf
<table>
<thead>
<tr>
<th>Section</th>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5</td>
<td>Another Application: Adding Integers</td>
<td>29</td>
</tr>
<tr>
<td>2.6</td>
<td>Arithmetic</td>
<td>33</td>
</tr>
<tr>
<td>2.7</td>
<td>Decision Making: Equality and Relational Operators</td>
<td>34</td>
</tr>
<tr>
<td>2.8</td>
<td>Wrap-Up</td>
<td>37</td>
</tr>
<tr>
<td>3</td>
<td>Introduction to Classes, Objects, Methods and Strings</td>
<td>38</td>
</tr>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>39</td>
</tr>
<tr>
<td>3.2</td>
<td>Instance Variables, <em>set</em> Methods and <em>get</em> Methods</td>
<td>39</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Account Class with an Instance Variable, a <em>set</em> Method and a <em>get</em> Method</td>
<td>40</td>
</tr>
<tr>
<td>3.2.2</td>
<td>AccountTest Class That Creates and Uses an Object of Class Account</td>
<td>43</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Compiling and Executing an App with Multiple Classes</td>
<td>46</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Account UML Class Diagram with an Instance Variable and <em>set</em> and <em>get</em> Methods</td>
<td>46</td>
</tr>
<tr>
<td>3.2.5</td>
<td>Additional Notes on Class AccountTest</td>
<td>47</td>
</tr>
<tr>
<td>3.2.6</td>
<td>Software Engineering with <em>private</em> Instance Variables and <em>public</em> <em>set</em> and <em>get</em> Methods</td>
<td>48</td>
</tr>
<tr>
<td>3.3</td>
<td>Primitive Types vs. Reference Types</td>
<td>49</td>
</tr>
<tr>
<td>3.4</td>
<td>Account Class: Initializing Objects with Constructors</td>
<td>50</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Declaring an Account Constructor for Custom Object Initialization</td>
<td>50</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Class AccountTest: Initializing Account Objects When They’re Created</td>
<td>51</td>
</tr>
<tr>
<td>3.5</td>
<td>Account Class with a Balance; Floating-Point Numbers</td>
<td>53</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Account Class with a <em>balance</em> Instance Variable of Type <em>double</em></td>
<td>54</td>
</tr>
<tr>
<td>3.5.2</td>
<td>AccountTest Class to Use Class Account</td>
<td>55</td>
</tr>
<tr>
<td>3.6</td>
<td>Wrap-Up</td>
<td>58</td>
</tr>
<tr>
<td>4</td>
<td>Control Statements: Part 1; Assignment, ++ and -- Operators</td>
<td>59</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>60</td>
</tr>
<tr>
<td>4.2</td>
<td>Control Structures</td>
<td>60</td>
</tr>
<tr>
<td>4.3</td>
<td>if Single-Selection Statement</td>
<td>62</td>
</tr>
<tr>
<td>4.4</td>
<td>if...else Double-Selection Statement</td>
<td>63</td>
</tr>
<tr>
<td>4.5</td>
<td>Student Class: Nested if...else Statements</td>
<td>67</td>
</tr>
<tr>
<td>4.6</td>
<td>while Repetition Statement</td>
<td>69</td>
</tr>
<tr>
<td>4.7</td>
<td>Counter-Controlled Repetition</td>
<td>71</td>
</tr>
<tr>
<td>4.8</td>
<td>Sentinel-Controlled Repetition</td>
<td>74</td>
</tr>
<tr>
<td>4.9</td>
<td>Nested Control Statements</td>
<td>79</td>
</tr>
<tr>
<td>4.10</td>
<td>Compound Assignment Operators</td>
<td>81</td>
</tr>
<tr>
<td>4.11</td>
<td>Increment and Decrement Operators</td>
<td>81</td>
</tr>
<tr>
<td>4.12</td>
<td>Primitive Types</td>
<td>84</td>
</tr>
<tr>
<td>4.13</td>
<td>Wrap-Up</td>
<td>85</td>
</tr>
</tbody>
</table>
## Contents

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

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
   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
   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
   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**

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
## Contents

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 1 332

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
   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
   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

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

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
Contents

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 New and Runnable States 657
  20.2.2 Waiting State 657
  20.2.3 Timed Waiting State 657
  20.2.4 Blocked State 657
  20.2.5 Terminated State 657
  20.2.6 Operating-System View of the Runnable 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: ArrayBlockingQueue 680
20.7 (Advanced) Producer/Consumer Relationship with synchronized, wait, notify and notifyAll 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: SwingWorker 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 sort/parallelSort Timings with the Java SE 8 Date/Time API 721
20.13 Java SE 8: Sequential vs. Parallel Streams 723
20.14 (Advanced) Interfaces Callable and Future 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 1: Object-Oriented Design with the UML  815
23.1 Case Study Introduction  816
Contents

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
<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>J Number Systems 963</td>
</tr>
<tr>
<td>J.1 Introduction 964</td>
</tr>
<tr>
<td>J.2 Abbreviating Binary Numbers as Octal and Hexadecimal Numbers 967</td>
</tr>
<tr>
<td>J.3 Converting Octal and Hexadecimal Numbers to Binary Numbers 968</td>
</tr>
<tr>
<td>J.4 Converting from Binary, Octal or Hexadecimal to Decimal 968</td>
</tr>
<tr>
<td>J.5 Converting from Decimal to Binary, Octal or Hexadecimal 969</td>
</tr>
<tr>
<td>J.6 Negative Binary Numbers: Two’s Complement Notation 971</td>
</tr>
<tr>
<td>K Bit Manipulation 973</td>
</tr>
<tr>
<td>K.1 Introduction 973</td>
</tr>
<tr>
<td>K.2 Bit Manipulation and the Bitwise Operators 973</td>
</tr>
<tr>
<td>K.3 BitSet Class 983</td>
</tr>
<tr>
<td>L Labeled break and continue Statements 987</td>
</tr>
<tr>
<td>L.1 Introduction 987</td>
</tr>
<tr>
<td>L.2 Labeled break Statement 987</td>
</tr>
<tr>
<td>L.3 Labeled continue Statement 988</td>
</tr>
<tr>
<td>M UML 2: Additional Diagram Types 990</td>
</tr>
<tr>
<td>M.1 Introduction 990</td>
</tr>
<tr>
<td>M.2 Additional Diagram Types 990</td>
</tr>
<tr>
<td>N Design Patterns 992</td>
</tr>
<tr>
<td>N.1 Introduction 992</td>
</tr>
<tr>
<td>N.2 Creational, Structural and Behavioral Design Patterns 993</td>
</tr>
<tr>
<td>N.2.1 Creational Design Patterns 994</td>
</tr>
<tr>
<td>N.2.2 Structural Design Patterns 996</td>
</tr>
<tr>
<td>N.2.3 Behavioral Design Patterns 997</td>
</tr>
<tr>
<td>N.2.4 Conclusion 998</td>
</tr>
<tr>
<td>N.3 Design Patterns in Packages java.awt and javax.swing 998</td>
</tr>
<tr>
<td>N.3.1 Creational Design Patterns 999</td>
</tr>
<tr>
<td>N.3.2 Structural Design Patterns 999</td>
</tr>
<tr>
<td>N.3.3 Behavioral Design Patterns 1001</td>
</tr>
<tr>
<td>N.3.4 Conclusion 1005</td>
</tr>
<tr>
<td>N.4 Concurrency Design Patterns 1005</td>
</tr>
<tr>
<td>N.5 Design Patterns Used in Packages java.io and java.net 1006</td>
</tr>
<tr>
<td>N.5.1 Creational Design Patterns 1006</td>
</tr>
<tr>
<td>N.5.2 Structural Design Patterns 1006</td>
</tr>
<tr>
<td>N.5.3 Architectural Patterns 1008</td>
</tr>
<tr>
<td>N.5.4 Conclusion 1010</td>
</tr>
<tr>
<td>N.6 Design Patterns Used in Package java.util 1010</td>
</tr>
</tbody>
</table>
This page intentionally left blank
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
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
diteit@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](http://www.deitel.com/deitelfan))
- Twitter® (@deitel)
- YouTube® ([http://youtube.com/DeitelTV](http://youtube.com/DeitelTV))
- LinkedIn® ([http://linkedin.com/company/deitel-&-associates](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—filter, map and reduce
- Library enhancements to support lambdas (e.g., java.util.stream, java.util.function)
- Date & Time API (java.time)
- Java concurrency API improvements
- static and default methods in interfaces
- Functional interfaces—interfaces that define only one abstract method and can include static and default 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.

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

**Fig. 4.2** | Java SE 8 lambdas and streams discussions and examples. (Part 1 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:


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

**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:


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](http://www.deitel.com/books/javafp3)
- [www.pearsonhighered.com/deitel](http://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.


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

This page intentionally left blank
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:


*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:


**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:


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 characterstypically 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 pro-
gramming. 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:


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

[https://netbeans.org/downloads/](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/](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:


The free version supports only Java SE development.
Before You Begin

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!
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.
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 predefined 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.

```java
// Fig. 3.1: Account.java
// Account class that contains a name instance variable
// and methods to set and get its value.

public class Account {
    private String name; // instance variable

    // method to set the name in the object
    public void setName(String name) {
        this.name = name; // store the name
    }

    // method to retrieve the name from the object
    public String getName() {
        return name; // return value of name to caller
    }
}
```

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
3.2  Instance Variables, set Methods and get Methods

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

```java
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):

```java
public void setName(String name) {
    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
}
```

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 Instance Variables, set Methods and get Methods

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.” 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.

```java
// Fig. 3.2: AccountTest.java
// Creating and manipulating an Account object.
import java.util.Scanner;

public class AccountTest {
    public static void main(String[] args) {
        // create a Scanner object to obtain input from the command window
        Scanner input = new Scanner(System.in);

        // create an Account object and assign it to myAccount
        Account myAccount = new Account();

        // display initial value of name (null)
        System.out.printf("Initial name is: %s\n\n", myAccount.getName());

        // prompt for and read name
        System.out.println("Please enter the name:");
        String theName = input.nextLine(); // read a line of text
        myAccount.setName(theName); // put theName in myAccount
        System.out.println(); // outputs a blank line

        // display the name stored in object myAccount
        System.out.printf("Name in object myAccount is: %s\n\n", myAccount.getName());
    }
} // end class AccountTest
```

Fig. 3.2  Creating and manipulating an Account object. (Part 1 of 2.)
Chapter 3 Introduction to Classes, Objects, Methods and Strings

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 whitespace 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 `println`, `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 (.),

```
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.)
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.” 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.
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
Chapter 3  Introduction to Classes, Objects, Methods and Strings

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
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.
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

```java
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
Objects that are referenced may each contain many instance variables. Line 10 of Fig. 3.2:

```java
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:

```java
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:

```java
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.)
3.4 Account Class: Initializing Objects with Constructors

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 public class AccountTest
5 {
6     public static void main(String[] args)
7     {
8         // create two Account objects
9         Account account1 = new Account("Jane Green");
10        Account account2 = new Account("John Blue");
11
12        // display initial value of name for each Account
13        System.out.printf("account1 name is: %s%n", account1.getName());
14        System.out.printf("account2 name is: %s%n", account2.getName());
15    }
16 } // 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.

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.

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.

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.

```
// Fig. 3.8: Account.java
// Account class with a double instance variable balance and a constructor and deposit method that perform validation.

public class Account
{
    private String name; // instance variable
    private double balance; // instance variable

    // Account constructor that receives two parameters
    public Account(String name, double balance)
    {
        this.name = name; // assign name to instance variable name
        // validate that the balance is greater than 0.0; if it's not,
        // instance variable balance keeps its default initial value of 0.0
        if (balance > 0.0) // if the balance is valid
            this.balance = balance; // assign it to instance variable balance
    }

    // method that deposits (adds) only a valid amount to the balance
    public void deposit(double depositAmount)
    {
        if (depositAmount > 0.0) // if the depositAmount is valid
            balance = balance + depositAmount; // add it to the balance
    }

    // method returns the account balance
    public double getBalance()
    {
        return balance;
    }

    // method that sets the name
    public void setName(String name)
    {
        this.name = name;
    }

    // method that returns the name
    public String getName()
    {
```

Fig. 3.8 | Account class with a double instance variable balance and a constructor and deposit method that perform validation. (Part 1 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.
public class AccountTest
{
    public static void main(String[] args)
    {
        Account account1 = new Account("Jane Green", 50.00);
        Account account2 = new Account("John Blue", -7.53);

        // display initial balance of each object
        System.out.printf("%s balance: $%.2fn", account1.getName(), account1.getBalance());
        System.out.printf("%s balance: $%.2fn", account2.getName(), account2.getBalance());

        // create a Scanner to obtain input from the command window
        Scanner input = new Scanner(System.in);

        System.out.print("Enter deposit amount for account1: "); // prompt
        double depositAmount = input.nextDouble(); // obtain user input
        System.out.printf("Adding %.2f to account1 balance\n\n", depositAmount);
        account1.deposit(depositAmount); // add to account1's balance

        // display balances
        System.out.printf("%s balance: $%.2fn", account1.getName(), account1.getBalance());
        System.out.printf("%s balance: $%.2fn", account2.getName(), account2.getBalance());

        System.out.print("Enter deposit amount for account2: "); // prompt
        depositAmount = input.nextDouble(); // obtain user input
        System.out.printf("Adding %.2f to account2 balance\n\n", depositAmount);
        account2.deposit(depositAmount); // add to account2 balance

        // display balances
        System.out.printf("%s balance: $%.2fn", account1.getName(), account1.getBalance());
        System.out.printf("%s balance: $%.2fn", account2.getName(), account2.getBalance());
    } // end main
} // 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`.

```
Account
- name : String
- balance : double
«constructor» Account(name : String, balance: double)
+ deposit(depositAmount : double)
+ getBalance() : double
+ setName(name : String)
+ getName() : String
```

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
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
action key 428
action in the UML 61, 638
action state symbol 116
action state in the UML 61
action of an object 61
action of an object 836
ActionListener 348, 353
ActionEvent class 348, 349, 353, 400
ActionEvent class (JavaFX) 809, 811
ActionEvent 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
active type arguments 589
cyclic 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
JMenu 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
addItemListener method of class Component 384
addLast method of LinkedList 518
addMouseListener method of interface ObservableValue 812
addListSelectionListener method of class JList 369
addMouseMotionListener 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
array initializer
array-creation expression
array-access expression
arithmetic overflow
argument index 946, 952, 960
argument list 947
argument promotion 125
argument to a method 45
argument to a method 81
argument types 589
arithmetic compound assignment operators 81
arithmetic operators 33
arithmetic overflow 74, 314
ArithmeticException class 232, 307, 313
array 145, 475, 1011
array-access expression 146
array-creation expression 147
array initializer 149
for multidimensional array 173
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
UnaryOperator (Java SE 8) 558
applyAsInt method of interface
UnaryOperator (Java SE 8) 560
arc 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
argument to a method 81
argument types 589
arithmetic compound assignment operators 81
arithmetic operators 33
arithmetic overflow 73, 314
ArithmeticException class 232, 307, 313
array 145, 475, 1011
bounds checking 159
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
array of one-dimensional arrays 173
ArrayBlockingQueue class 622, 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
block increment of a class 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 exclusive OR, | 112
Boolean primitive type 66, 898, 899, 906
border of a JFrame 616
BorderLayout class 377, 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 IntStream (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 598
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
BufferedInputOutputStream 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-instance call 166
call-by-value 166
call1 method of interface Callable 726
Callable interface 726
call1 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-throw requirement 316
cd to change directories 24
cell 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
collection directories 24
collection 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 652
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
caratterValue 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
Index 1017

Character class (cont.)
tToLowerCase method 460
tToUpperCase method 460
static conversion methods 461
tToLowerCase method 461
tToUpperCase method 460
character encoding 937
character set 938
character string 24
CharArrayReader class 506
CharArrayWriter class 506
class 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
ArithmeticException 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
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
CopyOnWriteArrayList 706
DataInputStream 504
DataOutputStream 504
Date 951
DelayQueue 706
Double 510, 606
DoubleProperty (JavaFX) 812
DriverManager 752
Classes (cont.)
E111ps2D 403
E111ps2D.Double 428
E111ps2D.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
FontFormat 477, 945
Frame 616
FXMLDocument (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
HashSet 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.)

JMenultem 618, 635
JOptionPane 335
JPanel 338, 381, 388, 397, 613
JPasswordField 344, 349
PopupMenu 625
JProgressBar 717
JRadioButton 357, 360, 363
JRadioButtonMenuItem 617, 618, 625
JScrollPane 369, 372, 400, 401
JTable 506
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 504
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, 384
MouseWheelEvent 374
Node (JavaFX) 790
Number 606
NumberFormat 230, 722, 800, 808
Object 216
ObjectInputStream 492
ObjectOutputStream 492
Optional (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
RuntimeManager 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
TableModel 532
TableModelEvent 766
TableRow 766
TextField (JavaFX) 800
TexturePaint 403, 431, 432
Throwables 314, 324
TreeMap 537
TreeSet 534
Types 754
UIDesigner 629
UnsupportedOperationExceptionException 517
Vbox (JavaFX) 795
Vector 511
Window 616, 617
WindowAdapter 378, 766
Writer 505

class-instance creation expression 813
ClassName through 623
CLASSPATH

environment variable 25, 227
classpath 227, 751

-classpath command-line argument
to java 228
to javac 227

CLASSPATH problem 13
clear debug 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

class client code 267
client tier 1009
close method of Object 262
closing 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 ObjectInputStream 497
of ResultSet 754
of Statement 754

close method of interface
AutoCloseable 330
closed polygons 425
ClosePath method of class
GeneralPath 435
closure 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
copy method 519, 526
disjoint method 519, 529
diff 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, diff, 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
color method 407, 409
color method 407
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
i 953
j 953
k 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
e 951
F 950
H 951
i 951
j 951
k 951
T 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
cordinate system 403, 405
coordinates (0, 0) 403
copy method of Collections 519, 526
copying files 477
copying objects
dee copy 262
shallowny 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-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
creating and initializing an array 148
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
cylic gradient 431
DataOutputStream 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 format 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
deadline mode 903
breakpoint 901
clear command 913
cnt command 904
define 901
default 909
-g compiler option 902
inserting breakpoints 903
jdbc 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
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
Decorating 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 464
delimiter 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-11st.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
diabetic 939
dialog 335
dialog box 335, 623
Dialog font 414
DialogInput font 414
diamond in the UML 61
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 ResultSet 767
disjoint method of 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 sign ($) 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 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 Graphics 421
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
toM1111s 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
class BorderLayout 377, 391
class GridBagConstraints 643
echo character of class JPasswordField
344
Eclipse
demonstration video
(www.deitel.com/books/
jhtp9) 21
Eclipse (www.eclipse.org) 9
Eclipse Foundation 7
et 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
eled UML diagram 826
elible for garbage collection 220
elive resource leaks 318
Eclipse20 class 403
Ellipse2D. Double class 428
Ellipse2D. Float class 428
ellipse (...) 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
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  
selAlignment 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 interface 378  
    name 414  
font (cont.)  
    size 414  
    style 414  
Font class 359, 403, 414  
  BOLD constant 414  
  getFamily method 413, 416  
  getName 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 (cont.)  
  %n (line separator) 29  
  %d 947  
  %f 949  
  %d 947  
  %9d, 946, 949  
  %9d 947  
  %9d 947  
format string 28, 946, 955  
formatted output 952  
  , (comma) formatting flag 96  
  %f formatting flag 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  
FORM 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
generics (cont.)
  wildcard without an upper bound 609
  wildcards 605, 607
  gesture 8
  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
  getColumn method of class
  Object 614
  getColumn method of class
  Object 614
  getColumn method of interface
  Method 262, 287
  getClassName method of class
  StackTraceElement 324
  getClassName method of class
  UIManager.LookAndFeelInfo 632
  getClickCount method of class
  MouseListener 380
  getColor method of class Color 407
  getColor method of class Graphics 407
  getColumnType method of class
  TableModel 755, 760
  getColumns method of class
  TableModel 755, 760
  getColumns method of class
  TableModel 755, 760
  getColumn method of interface
  Method 262, 287
  getSelected method of class
  FontMetrics 417
  getFont 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 class
  Calendar 952
  getIntersect method of class
  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
  getLatest 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
  getObjectName 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
  getPropText 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
  getSelectedValue method of class
  JList 372
  getSize method of class
  Font 413, 414
  getSource method of class
  EventObject 349
Interfaces (cont.)

Predicate 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) 554
IntConsumer functional interface (Java SE 8) 554
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
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
MouseListener 353, 372, 377, 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
Stream 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
resizeable 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)
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
applyAsInt method 560
IOException class
726
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
directory method of class Files
478
isEmpty method
NSArray 210
Map 541
Stack 533
isItalic method of class Font
413, 416
isJavaIdentifierPart method of class
Character 460
isSerializable method of class
Character 460
isLetter method of class
Character 460
isLetterOrDigit method of class
Character 460
isLowerCase method of class Character
460
isUpperCase method of class Character
460
JSObject method of class
Object 380, 387
JSObject method of class Font
413, 416
JSObject method of class SwingWorker
628
JSObject method of class
AbstractButton 625
JCheckBox 360
JSObject method of class
InputEvent 387
JSObject method of class Character
460
ITrust constant of class Font
414
ItemEvent class 360, 363
generateChange method 367
Java SE 7 (Java Standard Edition 7)
Java programming language 7
Java Language Package
Java Keywords 898
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 inner classes with lambdas 567
   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
   implementing event listeners with lambdas 350, 612
   Instant class 729
   IntBinaryOperator functional interface 558
   IntConsumer functional interface 556
   IntFunction functional interface 377
   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
   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.imagedata 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 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
   java.util.Calendar class 951
   java.util.List class 127
   java.util.concurrent package 698, 699
   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/"
   3Jdk/33
   Java2D API 428
   javac compiler 10, 24
   javadeprecation flag 30
   javadoc comment 22
   javadoc options
   -author 933
   -d 933
   -link 933
    ItemLister interface 360, 624
    itemStateChanged method 360, 625
    iterator 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
    iteration 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
KeyEvent
KeyAdapter
key-value pair
Key event handling
key constant
JToggleButton
JTextField
JTextField
JTable
JTabbedPane
JSlider
JScrollPane
1030
just-in-time (JIT) compiler
keyboard 29, 333, 788
647
400
398
345
isActionKey
addActionListener
setText
setEditable
setDisabledTextColor
text
getSelectedText
setRowSorter
getRowSorter
setRowFilter
setRowFilter method
setRowSorter method
TableRowSorter
setRowSorter method
JTextField class 338, 344, 352, 398
addActionListener method
JTextField and JPasswordField
JToggleButton class 357
just-in-time compilation
just-in-time (JIT) compiler
K
key constant
key event
Key event handling
key–value pair
KeyAdapter class
keyboard 29, 333, 788
KeyEvent class 353, 384
getKeyChar method
getKeyCode method
getKeyModifiersText method
getKeyText method
isActionKey method
KeyListener interface 353, 378, 384
keyPressed method 384, 387
keyReleased method 384
keyTyped method 384
Keypad class
getKeyText
getKeyModifiersText
getKeyCode
getKeyChar
isActionKey
L
label in a switch
labeled block
labeled break statement
exiting a nested for statement
labeled continue statement
terminating a single iteration of a labeled-for statement
labeled statement
labels for tick marks
lambda (Java SE 8)
lambda expression
composing
lambda expression (Java SE 8)
lambda expressions
arrow token (->)
event handler
method references
parameter list
target type
with an empty parameter list
implementing an event handler
lambda expressions
statement block
type inference
LAMP
language package
last-in, first-out (LIFO) order
language package
LAMP
Layers architecture pattern
layout containers
layout manager
BorderLayout
FlowLayout
GridLayout
LayoutManager
LayoutContainer method of interface
LayoutManager
LayoutManager interface
LayoutManager method
LayoutManager2 interface
Layouts
GridPane
Vbox
lazy
lazy quantifier
lazy stream operation (Java SE 8)
Lea, Doug
leading
leaf
left brace
LEFT constant of class
left justification
left justified
left justified
left-mouse-button click
left shift
Left, center and right mouse-button clicks
length field of an array
length instance variable of an array
length method of class
String
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 351
minor 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
MouseAdapter 373, 378
mouseDragged method of interface
MouseMotionListener 373, 381
mouseEntered method of interface
MouseListener 373
MouseEvent class 353, 372, 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
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
multiply
multiplicative operators:
multiplication compound assignment 33
multiple-selection list 367
multiple-selection list 367
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
class 194, 347, 629
relationship between an inner class
and its top-level class 360
Nest class
Nest Class Summary section in API 919
nest control statement 988
nest control statements 79, 132
Examination-results problem 79
nest for statement 154, 174, 175, 176, 180, 987
enhanced for 175
nest if selection statement 67
nest 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
newCachedThreadPool method of class
Executors 661
newCondition 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 file, 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
NumberFormatException exception 163
NumberFormatException 606
doubleValue method 607
number systems 461
NumberFormatException class 230, 722, 800, 808
format method 231, 722
currencyInstance method 231
getCurrencyInstance method 722, 809
setRoundingMode method 812
numeric Classes 510

O

object 2, 4
object (or instance) 848
Object class 216, 235, 239, 500
class 216, 235, 239, 500
class (Java SE 8) 556
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
ObjectOutputStream interface 492
writeObject method 493
ObjectNameOutputStream class 492, 492, 493, 543, 1006
close method 497
Observable class 1003
ObservableValue interface 809
ObservableValue interface (JavaFX) 809
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 PriorityBlockingQueue 533
ON clause 744
ONE constant of class BigDecimal 231
to-one relationship 738, 738
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
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 file 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
operator 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 81
/,, 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
++, addition assignment operator 81
= =, subtraction assignment operator 81
|, logical inclusive OR 110, 112
||, logical inclusive OR 110, 111
|, logical OR 110, 111
|, 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
gas method 557, 575
orElse method 557, 575
or method of class BitSet 984
or method of functional interface
Predicate (Java SE 8) 563
polymorphically process Invokes and Employees 299
polymerism 104, 263, 265, 862, 863, 873
preemptive scheduling
predicate method 105, 740
predicate 559
predecrement
precondition
precision 946, 947
Precedence Chart Appendix 895
33
preference
power of 2 larger than 100 69
power (exponent) 120
PreparedStatement
Pref Width
Pref Height
Predicate
pow
pow
pop
polymorphically process
polymorphism 104, 263, 265, 862, 863, 873
preorder
precendent
predicates
preincrementing and postincrementing
preincrement
prefix increment operator
preferred size (JavaFX) 796
Preferences API
prefix decrement operator
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 720
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
generate method 541
keySet method 544
last method 543
load method 544
setProperty method 541
store method 543
propertyChange method of interface
PropertyChangeListener interface
PropertyChangeListener 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
QUERY_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
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
SQLExceptionNotSupported-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
ol package java.util 531
peek method 533
pop method 533
push method 532
Stack generic class 594
StackDouble> 601
Stack<Integer> 601
Stack generic class declaration 595
stack trace 307
stack unwinding 322
StackTraceElement class 324
class searching methods 445
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
tstate 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
do...while 62, 97, 98
double selection 66
double selection 66
enhanced for 163
for 62, 64, 66, 91, 92, 94, 95
if 34, 61, 62, 63, 98
if...else 61, 63, 64, 75, 98
if...else 61, 63, 64, 75, 98
if...else 61, 63, 64, 75, 98
if...else 61, 63, 64, 75, 98
if...else 61, 63, 64, 75, 98
if...else 61, 63, 64, 75, 98
if...else 61, 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
key word 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
forEachEach method 563
map method 565, 565
sorted method 563, 566
Stream interface (Java SE 8) 552
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
Stream class 437
charAt method 439, 454
comeTo method 441, 443
concat method 448
decidedWith 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 method 517
toLowerCase method 450
toUpperCase method 517
toUpperCase method 449
trim method 450
valueOf method 450
String class searching methods 445
thenComparing method of functional interface Comparator (Java SE 8) 571

throw an exception 419

three-dimensional rectangle 419

three-button mouse 380

thread 312, 405, 1005

cycle 656, 658

execution 655

scheduling 658, 677

state 656

synchronization 544, 664

Thread
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 655

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

two-dimensional rectangle 419

throw an exception 157, 306, 310

throw an exception 196, 206

throw keyword 321, 898

throw point 308

throw statement 320

Thrower class 314, 324

getLocalizedMessage 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

tick 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 buttons 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

toNanos 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

transacted 835, 838

transformation 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 reserved word 63, 66

truncate 33

truncate fractional part of a calculation 73

truncate 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 799, 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 new notation (Java

SE 7) 514

type inferring 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 Color 458

TYPE_SCROLL_INSENSITIVE constant 759

TYPE_SCROLL_SENSITIVE constant 759

Types class 754

typesetting system 437

type wraper 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
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
  uneditable 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 517
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
URL (Uniform Resource Locator) 827
URIBuilder class 1007
URL (Uniform Resource Identifier) 478
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 an interface ListSelectionListener 369
valueOf method of class BigDecimal 230
valueOf method of class String 450
valueOf method of an enum 214
VALUES SQL clause 745
  variable 31
  name 31
  reference type 49
  variable declaration statement 31
  variable is not modifiable 221
  variable-length argument list 182
  variable names
  camel case naming 40
  variable scope 90
VBox class (JavaFX) 795
  alignment 796
  Alignment property 796
Vector class 511
  verb phrase in requirements document 839
@version javadoc tag 932
Version
  note 932
VIRTUAL constant of class
  GridBagConstraints 644
  vertical gap space 393
  vertical scrolling 400
  vertical strut 641
VIRTUAL_SCROLLBAR_ALWays constant of class
  JScrollPane 401
VIRTUAL_SCROLLBAR_A5_Needed constant of class
  JScrollPane 401
VIRTUAL_SCROLLBAR_NEVER constant of class
  JScrollPane 401
Vgap property of a GridPane 805
  v1 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
  weightx 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
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 relationship 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 gadgets 333, 788
windowActivated method of interface
WindowListener 617
WindowAdapter class 378, 766
windowClosed method of interface
WindowListener 617, 766
windowClosing method 617
windowDeactivated method 617
windowDeiconified method 617
windowIconified method of interface
WindowListener 617
window 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
Withdrawal 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 505
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