Java for Programmers, Second Edition

Contains 200+ Examples
Covers Java SE 7

Paul Deitel • Harvey Deitel
Trademarks
DEITEL, the double-thumbs-up bug and DIVE INTO are registered trademarks of Deitel and Associates, Inc.
Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.
Microsoft, Internet Explorer and the Windows logo are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
UNIX is a registered trademark of The Open Group.
Apache is a trademark of The Apache Software Foundation.
CSS, XHTML and XML are registered trademarks of the World Wide Web Consortium.
Firefox is a registered trademark of the Mozilla Foundation.
Google is a trademark of Google, Inc.
Web 2.0 is a service mark of CMP Media.

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.
In memory of Clifford “Spike” Stephens,
A dear friend who will be greatly missed.

Paul and Harvey Deitel
This page intentionally left blank
Preface

Before You Begin

1 Introduction
1.1 Introduction 2
1.2 Introduction to Object Technology 2
1.3 Open Source Software 5
1.4 Java and a Typical Java Development Environment 7
1.5 Test-Driving a Java Application 11
1.6 Web 2.0: Going Social 15
1.7 Software Technologies 18
1.8 Keeping Up to Date with Information Technologies 20
1.9 Wrap-Up 21

2 Introduction to Java Applications
2.1 Introduction 23
2.2 Your First Program in Java: Printing a Line of Text 23
2.3 Modifying Your First Java Program 27
2.4 Displaying Text with printf 29
2.5 Another Application: Adding Integers 30
2.6 Arithmetic 34
2.7 Decision Making: Equality and Relational Operators 35
2.8 Wrap-Up 38

3 Introduction to Classes, Objects, Methods and Strings
3.1 Introduction 40
3.2 Declaring a Class with a Method and Instantiating an Object of a Class 40
3.3 Declaring a Method with a Parameter 44
3.4 Instance Variables, set Methods and get Methods 47
3.5 Primitive Types vs. Reference Types 52
3.6 Initializing Objects with Constructors 53
## Contents

3.7 Floating-Point Numbers and Type `double` 56  
3.8 Wrap-Up 60  

### 4 Control Statements: Part 1 61  
4.1 Introduction 62  
4.2 Control Structures 62  
4.3 `if` Single-Selection Statement 64  
4.4 `if`…`else` Double-Selection Statement 65  
4.5 `while` Repetition Statement 68  
4.6 Counter-Controlled Repetition 70  
4.7 Sentinel-Controlled Repetition 73  
4.8 Compound Assignment Operators 81  
4.10 Increment and Decrement Operators 82  
4.11 Primitive Types 85  
4.12 Wrap-Up 85  

### 5 Control Statements: Part 2 86  
5.1 Introduction 87  
5.2 Essentials of Counter-Controlled Repetition 87  
5.3 `for` Repetition Statement 89  
5.4 Examples Using the `for` Statement 92  
5.5 `do`…`while` Repetition Statement 96  
5.6 `switch` Multiple-Selection Statement 98  
5.7 `break` and `continue` Statements 105  
5.8 Logical Operators 107  
5.9 Wrap-Up 113  

### 6 Methods: A Deeper Look 114  
6.1 Introduction 115  
6.2 Program Modules in Java 115  
6.3 `static` Methods, `static` Fields and Class `Math` 115  
6.4 Declaring Methods with Multiple Parameters 118  
6.5 Notes on Declaring and Using Methods 121  
6.6 Argument Promotion and Casting 122  
6.7 Java API Packages 123  
6.8 Case Study: Random-Number Generation 125  
   6.8.1 Generalized Scaling and Shifting of Random Numbers 129  
   6.8.2 Random-Number Repeatability for Testing and Debugging 129  
6.9 Case Study: A Game of Chance; Introducing Enumerations 130  
6.10 Scope of Declarations 134  
6.11 Method Overloading 137  
6.12 Wrap-Up 139
## 7 Arrays and ArrayLists 140

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 Introduction</td>
<td>141</td>
</tr>
<tr>
<td>7.2 Arrays</td>
<td>141</td>
</tr>
<tr>
<td>7.3 Declaring and Creating Arrays</td>
<td>143</td>
</tr>
<tr>
<td>7.4 Examples Using Arrays</td>
<td>144</td>
</tr>
<tr>
<td>7.5 Case Study: Card Shuffling and Dealing Simulation</td>
<td>153</td>
</tr>
<tr>
<td>7.6 Enhanced for Statement</td>
<td>157</td>
</tr>
<tr>
<td>7.7 Passing Arrays to Methods</td>
<td>159</td>
</tr>
<tr>
<td>7.8 Case Study: Class GradeBook Using an Array to Store Grades</td>
<td>162</td>
</tr>
<tr>
<td>7.9 Multidimensional Arrays</td>
<td>167</td>
</tr>
<tr>
<td>7.10 Case Study: Class GradeBook Using a Two-Dimensional Array</td>
<td>171</td>
</tr>
<tr>
<td>7.11 Variable-Length Argument Lists</td>
<td>177</td>
</tr>
<tr>
<td>7.12 Using Command-Line Arguments</td>
<td>178</td>
</tr>
<tr>
<td>7.13 Class Arrays</td>
<td>180</td>
</tr>
<tr>
<td>7.14 Introduction to Collections and Class ArrayList</td>
<td>183</td>
</tr>
<tr>
<td>7.15 Wrap-Up</td>
<td>186</td>
</tr>
</tbody>
</table>

## 8 Classes and Objects: A Deeper Look 187

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1 Introduction</td>
<td>188</td>
</tr>
<tr>
<td>8.2 Time Class Case Study</td>
<td>188</td>
</tr>
<tr>
<td>8.3 Controlling Access to Members</td>
<td>192</td>
</tr>
<tr>
<td>8.4 Referring to the Current Object’s Members with the this Reference</td>
<td>193</td>
</tr>
<tr>
<td>8.5 Time Class Case Study: Overloaded Constructors</td>
<td>195</td>
</tr>
<tr>
<td>8.6 Default and No-Argument Constructors</td>
<td>201</td>
</tr>
<tr>
<td>8.7 Notes on Set and Get Methods</td>
<td>202</td>
</tr>
<tr>
<td>8.8 Composition</td>
<td>203</td>
</tr>
<tr>
<td>8.9 Enumerations</td>
<td>206</td>
</tr>
<tr>
<td>8.10 Garbage Collection and Method finalize</td>
<td>209</td>
</tr>
<tr>
<td>8.11 static Class Members</td>
<td>210</td>
</tr>
<tr>
<td>8.12 static Import</td>
<td>213</td>
</tr>
<tr>
<td>8.13 final Instance Variables</td>
<td>214</td>
</tr>
<tr>
<td>8.14 Time Class Case Study: Creating Packages</td>
<td>215</td>
</tr>
<tr>
<td>8.15 Package Access</td>
<td>221</td>
</tr>
<tr>
<td>8.16 Wrap-Up</td>
<td>222</td>
</tr>
</tbody>
</table>

## 9 Object-Oriented Programming: Inheritance 224

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1 Introduction</td>
<td>225</td>
</tr>
<tr>
<td>9.2 Superclasses and Subclasses</td>
<td>226</td>
</tr>
<tr>
<td>9.3 protected Members</td>
<td>228</td>
</tr>
<tr>
<td>9.4 Relationship between Superclasses and Subclasses</td>
<td>228</td>
</tr>
<tr>
<td>9.4.1 Creating and Using a CommissionEmployee Class</td>
<td>229</td>
</tr>
<tr>
<td>9.4.2 Creating and Using a BasePlusCommissionEmployee Class</td>
<td>235</td>
</tr>
<tr>
<td>9.4.3 Creating a CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy</td>
<td>240</td>
</tr>
</tbody>
</table>
10 Object-Oriented Programming: Polymorphism 254
10.1 Introduction 255
10.2 Polymorphism Examples 257
10.3 Demonstrating Polymorphic Behavior 258
10.4 Abstract Classes and Methods 260
10.5 Case Study: Payroll System Using Polymorphism 262
  10.5.1 Abstract Superclass Employee 263
  10.5.2 Concrete Subclass SalariedEmployee 266
  10.5.3 Concrete Subclass HourlyEmployee 268
  10.5.4 Concrete Subclass CommissionEmployee 270
  10.5.5 Indirect Concrete Subclass BasePlusCommissionEmployee 271
  10.5.6 Polymorphic Processing, Operator instanceof and Downcasting 273
  10.5.7 Summary of the Allowed Assignments Between Superclass and Subclass Variables 277
10.6 final Methods and Classes 278
10.7 Case Study: Creating and Using Interfaces 279
  10.7.1 Developing a Payable Hierarchy 280
  10.7.2 Interface Payable 281
  10.7.3 Class Invoice 282
  10.7.4 Modifying Class Employee to Implement Interface Payable 284
  10.7.5 Modifying Class SalariedEmployee for Use in the Payable Hierarchy 286
  10.7.6 Using Interface Payable to Process Invoices and Employees Polymorphically 288
  10.7.7 Common Interfaces of the Java API 289
10.8 Wrap-Up 290

11 Exception Handling: A Deeper Look 292
11.1 Introduction 293
11.2 Example: Divide by Zero without Exception Handling 293
11.3 Example: Handling ArithmeticExceptions and InputMismatchExceptions 296
11.4 When to Use Exception Handling 301
11.5 Java Exception Hierarchy 301
11.6 finally Block 304
11.7 Stack Unwinding and Obtaining Information from an Exception Object 308
11.8 Chained Exceptions 311
11.9 Declaring New Exception Types 313
11.10 Preconditions and Postconditions 314
11.11 Assertions 315
11.12 (New in Java SE 7) Multi-catch: Handling Multiple Exceptions in One catch 316
11.13 (New in Java SE 7) try-with-Resources: Automatic Resource Deallocation 316
11.14 Wrap-Up 317

12 ATM Case Study, Part 1: Object-Oriented Design with the UML 318
12.1 Case Study Introduction 319
12.2 Examining the Requirements Document 319
12.3 Identifying the Classes in a Requirements Document 327
12.4 Identifying Class Attributes 333
12.5 Identifying Objects’ States and Activities 338
12.6 Identifying Class Operations 342
12.7 Indicating Collaboration Among Objects 348
12.8 Wrap-Up 355

13 ATM Case Study Part 2: Implementing an Object-Oriented Design 359
13.1 Introduction 360
13.2 Starting to Program the Classes of the ATM System 360
13.3 Incorporating Inheritance and Polymorphism into the ATM System 365
13.4 ATM Case Study Implementation 371
13.4.1 Class ATM 372
13.4.2 Class Screen 377
13.4.3 Class Keypad 378
13.4.4 Class CashDispenser 379
13.4.5 Class DepositSlot 380
13.4.6 Class Account 381
13.4.7 Class BankDatabase 383
13.4.8 Class Transaction 386
13.4.9 Class BalanceInquiry 387
13.4.10 Class Withdrawal 388
13.4.11 Class Deposit 392
13.4.12 Class ATMCaseStudy 395
13.5 Wrap-Up 395

14 GUI Components: Part 1 398
14.1 Introduction 399
14.2 Java’s New Nimbus Look-and-Feel 400
14.3 Simple GUI-Based Input/Output with JOptionPane 401
14.4 Overview of Swing Components 404
14.5 Displaying Text and Images in a Window 406
14.6 Text Fields and an Introduction to Event Handling with Nested Classes 410
14.7 Common GUI Event Types and Listener Interfaces 416
14.8 How Event Handling Works 418
14.9 JButton 420
14.10 Buttons That Maintain State 423
  14.10.1 JCheckBox 423
  14.10.2 JRadioButton 426
14.11 JComboBox; Using an Anonymous Inner Class for Event Handling 429
14.12 JList 433
14.13 Multiple-Selection Lists 435
14.14 Mouse Event Handling 438
14.15 Adapter Classes 443
14.16 JPanel Subclass for Drawing with the Mouse 446
14.17 Key Event Handling 450
14.18 Introduction to Layout Managers 453
  14.18.1 FlowLayout 454
  14.18.2 BorderLayout 457
  14.18.3 GridLayout 460
14.19 Using Panels to Manage More Complex Layouts 462
14.20 JTextArea 464
14.21 Wrap-Up 467

15 **Graphics and Java 2D** 468
15.1 Introduction 469
15.2 Graphics Contexts and Graphics Objects 471
15.3 Color Control 472
15.4 Manipulating Fonts 479
15.5 Drawing Lines, Rectangles and Ovals 484
15.6 Drawing Arcs 488
15.7 Drawing Polygons and Polylines 491
15.8 Java 2D API 494
15.9 Wrap-Up 501

16 **Strings, Characters and Regular Expressions** 502
16.1 Introduction 503
16.2 Fundamentals of Characters and Strings 503
16.3 Class String 504
  16.3.1 String Constructors 504
  16.3.2 String Methods length, charAt and getChars 505
  16.3.3 Comparing Strings 506
  16.3.4 Locating Characters and Substrings in Strings 511
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.3.5</td>
<td>Extracting Substrings from Strings</td>
<td>513</td>
</tr>
<tr>
<td>16.3.6</td>
<td>Concatenating Strings</td>
<td>514</td>
</tr>
<tr>
<td>16.3.7</td>
<td>Miscellaneous String Methods</td>
<td>514</td>
</tr>
<tr>
<td>16.3.8</td>
<td>String Method <code>valueOf</code></td>
<td>516</td>
</tr>
<tr>
<td>16.4</td>
<td>Class <code>StringBuilder</code></td>
<td>517</td>
</tr>
<tr>
<td>16.4.1</td>
<td><code>StringBuilder</code> Constructors</td>
<td>518</td>
</tr>
<tr>
<td>16.4.2</td>
<td><code>StringBuilder</code> Methods <code>length</code>, <code>capacity</code>, <code>setLength</code> and <code>ensureCapacity</code></td>
<td>518</td>
</tr>
<tr>
<td>16.4.3</td>
<td><code>StringBuilder</code> Methods <code>charAt</code>, <code>setCharAt</code>, <code>getChars</code> and <code>reverse</code></td>
<td>520</td>
</tr>
<tr>
<td>16.4.4</td>
<td><code>StringBuilder</code> append Methods</td>
<td>521</td>
</tr>
<tr>
<td>16.4.5</td>
<td><code>StringBuilder</code> Insertion and Deletion Methods</td>
<td>523</td>
</tr>
<tr>
<td>16.5</td>
<td>Class <code>Character</code></td>
<td>524</td>
</tr>
<tr>
<td>16.6</td>
<td>Tokenizing Strings</td>
<td>529</td>
</tr>
<tr>
<td>16.7</td>
<td>Regular Expressions, Class <code>Pattern</code> and Class <code>Matcher</code></td>
<td>530</td>
</tr>
<tr>
<td>16.8</td>
<td>Wrap-Up</td>
<td>538</td>
</tr>
</tbody>
</table>

**17 Files, Streams and Object Serialization**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.1</td>
<td>Introduction</td>
<td>540</td>
</tr>
<tr>
<td>17.2</td>
<td>Files and Streams</td>
<td>540</td>
</tr>
<tr>
<td>17.3</td>
<td>Class <code>File</code></td>
<td>542</td>
</tr>
<tr>
<td>17.4</td>
<td>Sequential-Access Text Files</td>
<td>546</td>
</tr>
<tr>
<td>17.4.1</td>
<td>Creating a Sequential-Access Text File</td>
<td>546</td>
</tr>
<tr>
<td>17.4.2</td>
<td>Reading Data from a Sequential-Access Text File</td>
<td>553</td>
</tr>
<tr>
<td>17.4.3</td>
<td>Case Study: A Credit-Inquiry Program</td>
<td>556</td>
</tr>
<tr>
<td>17.4.4</td>
<td>Updating Sequential-Access Files</td>
<td>561</td>
</tr>
<tr>
<td>17.5</td>
<td>Object Serialization</td>
<td>562</td>
</tr>
<tr>
<td>17.5.1</td>
<td>Creating a Sequential-Access File Using Object Serialization</td>
<td>563</td>
</tr>
<tr>
<td>17.5.2</td>
<td>Reading and Deserializing Data from a Sequential-Access File</td>
<td>569</td>
</tr>
<tr>
<td>17.6</td>
<td>Additional <code>java.io</code> Classes</td>
<td>571</td>
</tr>
<tr>
<td>17.6.1</td>
<td>Interfaces and Classes for Byte-Based Input and Output</td>
<td>571</td>
</tr>
<tr>
<td>17.6.2</td>
<td>Interfaces and Classes for Character-Based Input and Output</td>
<td>573</td>
</tr>
<tr>
<td>17.7</td>
<td>Opening Files with <code>JFileChooser</code></td>
<td>574</td>
</tr>
<tr>
<td>17.8</td>
<td>Wrap-Up</td>
<td>577</td>
</tr>
</tbody>
</table>

**18 Generic Collections**

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.1</td>
<td>Introduction</td>
<td>579</td>
</tr>
<tr>
<td>18.2</td>
<td>Collections Overview</td>
<td>579</td>
</tr>
<tr>
<td>18.3</td>
<td>Type-Wrapper Classes for Primitive Types</td>
<td>580</td>
</tr>
<tr>
<td>18.4</td>
<td>Autoboxing and Auto-Unboxing</td>
<td>581</td>
</tr>
<tr>
<td>18.5</td>
<td>Interface <code>Collection</code> and Class <code>Collections</code></td>
<td>581</td>
</tr>
<tr>
<td>18.6</td>
<td>Lists</td>
<td>582</td>
</tr>
<tr>
<td>18.6.1</td>
<td><code>ArrayList</code> and <code>Iterator</code></td>
<td>583</td>
</tr>
<tr>
<td>18.6.2</td>
<td><code>LinkedList</code></td>
<td>585</td>
</tr>
</tbody>
</table>
## 21 Multimedia: Applets and Applications

- **21.1 Introduction**: 668
- **21.2 Loading, Displaying and Scaling Images**: 669
- **21.3 Animating a Series of Images**: 675
- **21.4 Image Maps**: 682
- **21.5 Loading and Playing Audio Clips**: 685
- **21.6 Playing Video and Other Media with Java Media Framework**: 688
- **21.7 Wrap-Up**: 692
- **21.8 Web Resources**: 692

## 22 GUI Components: Part 2

- **22.1 Introduction**: 695
- **22.2 JSlider**: 695
- **22.3 Windows: Additional Notes**: 699
- **22.4 Using Menus with Frames**: 700
- **22.5 JPopupMenu**: 708
- **22.6 Pluggable Look-and-Feel**: 711
- **22.7 JDesktopPane and JInternalFrame**: 716
- **22.8 JTabbedPane**: 720
- **22.9 Layout Managers: BoxLayout and GridBagLayout**: 722
- **22.10 Wrap-Up**: 734

## 23 Multithreading

- **23.1 Introduction**: 736
- **23.2 Thread States: Life Cycle of a Thread**: 738
- **23.3 Creating and Executing Threads with Executor Framework**: 741
- **23.4 Thread Synchronization**
  - **23.4.1 Unsynchronized Data Sharing**: 745
  - **23.4.2 Synchronized Data Sharing—Making Operations Atomic**: 749
- **23.5 Producer/Consumer Relationship without Synchronization**: 752
- **23.6 Producer/Consumer Relationship: ArrayBlockingQueue**: 760
- **23.7 Producer/Consumer Relationship with Synchronization**: 763
- **23.8 Producer/Consumer Relationship: Bounded Buffers**: 769
- **23.9 Producer/Consumer Relationship: The Lock and Condition Interfaces**: 776
- **23.10 Concurrent Collections Overview**: 783
- **23.11 Multithreading with GUI**
  - **23.11.1 Performing Computations in a Worker Thread**: 786
  - **23.11.2 Processing Intermediate Results with SwingWorker**: 792
- **23.12 Interfaces Callable and Future**: 799
- **23.13 Java SE 7: Fork/Join Framework**: 799
- **23.14 Wrap-Up**: 800
Contents

24 Networking

24.1 Introduction 802
24.2 Manipulating URLs 803
24.3 Reading a File on a Web Server 808
24.4 Establishing a Simple Server Using Stream Sockets 811
24.5 Establishing a Simple Client Using Stream Sockets 813
24.6 Client/Server Interaction with Stream Socket Connections 813
24.7 Datagrams: Connectionless Client/Server Interaction 825
24.8 Client/Server Tic-Tac-Toe Using a Multithreaded Server 833
24.9 [Web Bonus] Case Study: DeitelMessenger 848
24.10 Wrap-Up 848

25 Accessing Databases with JDBC

25.1 Introduction 850
25.2 Relational Databases 851
25.3 Relational Database Overview: The books Database 852
25.4 SQL
   25.4.1 Basic SELECT Query 856
   25.4.2 WHERE Clause 857
   25.4.3 ORDER BY Clause 859
   25.4.4 Merging Data from Multiple Tables: INNER JOIN 860
   25.4.5 INSERT Statement 862
   25.4.6 UPDATE Statement 863
   25.4.7 DELETE Statement 864
25.5 Instructions for Installing MySQL and MySQL Connector/J 864
25.6 Instructions for Setting Up a MySQL User Account 865
25.7 Creating Database books in MySQL 866
25.8 Manipulating Databases with JDBC
   25.8.1 Connecting to and Querying a Database 867
   25.8.2 Querying the books Database 872
25.9 RowSet Interface 885
25.10 Java DB/Apache Derby 887
25.11 PreparedStatements 889
25.12 Stored Procedures 904
25.13 Transaction Processing 905
25.14 Wrap-Up 905
25.15 Web Resources 906

26 JavaServer™ Faces Web Apps: Part I

26.1 Introduction 908
26.2 HyperText Transfer Protocol (HTTP) Transactions 909
26.3 Multitier Application Architecture 912
26.4 Your First JSF Web App
   26.4.1 The Default index.xhtml Document: Introducing Facelets 914
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>26.4.2</td>
<td>Examining the WebTimeBean Class</td>
</tr>
<tr>
<td>26.4.3</td>
<td>Building the WebTime JSF Web App in NetBeans</td>
</tr>
<tr>
<td>26.5</td>
<td>Model-View-Controller Architecture of JSF Apps</td>
</tr>
<tr>
<td>26.6</td>
<td>Common JSF Components</td>
</tr>
<tr>
<td>26.7</td>
<td>Validation Using JSF Standard Validators</td>
</tr>
<tr>
<td>26.8</td>
<td>Session Tracking</td>
</tr>
<tr>
<td>26.8.1</td>
<td>Cookies</td>
</tr>
<tr>
<td>26.8.2</td>
<td>Session Tracking with @SessionScoped Beans</td>
</tr>
<tr>
<td>26.9</td>
<td>Wrap-Up</td>
</tr>
<tr>
<td>27</td>
<td>JavaServer™ Faces Web Apps: Part 2</td>
</tr>
<tr>
<td>27.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>27.2</td>
<td>Accessing Databases in Web Apps</td>
</tr>
<tr>
<td>27.2.1</td>
<td>Setting Up the Database</td>
</tr>
<tr>
<td>27.2.2</td>
<td>@ManagedBean Class AddressBean</td>
</tr>
<tr>
<td>27.2.3</td>
<td>index.xhtml Facelets Page</td>
</tr>
<tr>
<td>27.2.4</td>
<td>addentry.xhtml Facelets Page</td>
</tr>
<tr>
<td>27.3</td>
<td>Ajax</td>
</tr>
<tr>
<td>27.4</td>
<td>Adding Ajax Functionality to the Validation App</td>
</tr>
<tr>
<td>27.5</td>
<td>Wrap-Up</td>
</tr>
<tr>
<td>28</td>
<td>Web Services</td>
</tr>
<tr>
<td>28.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>28.2</td>
<td>Web Service Basics</td>
</tr>
<tr>
<td>28.3</td>
<td>Simple Object Access Protocol (SOAP)</td>
</tr>
<tr>
<td>28.4</td>
<td>Representational State Transfer (REST)</td>
</tr>
<tr>
<td>28.5</td>
<td>JavaScript Object Notation (JSON)</td>
</tr>
<tr>
<td>28.6</td>
<td>Publishing and Consuming SOAP-Based Web Services</td>
</tr>
<tr>
<td>28.6.1</td>
<td>Creating a Web Application Project and Adding a Web Service Class in NetBeans</td>
</tr>
<tr>
<td>28.6.2</td>
<td>Defining the WelcomeSOAP Web Service in NetBeans</td>
</tr>
<tr>
<td>28.6.3</td>
<td>Publishing the WelcomeSOAP Web Service from NetBeans</td>
</tr>
<tr>
<td>28.6.4</td>
<td>Testing the WelcomeSOAP Web Service with GlassFish Application Server's Tester Web Page</td>
</tr>
<tr>
<td>28.6.5</td>
<td>Describing a Web Service with the Service Description Language (WSDL)</td>
</tr>
<tr>
<td>28.6.6</td>
<td>Creating a Client to Consume the WelcomeSOAP Web Service</td>
</tr>
<tr>
<td>28.6.7</td>
<td>Consuming the WelcomeSOAP Web Service</td>
</tr>
<tr>
<td>28.7</td>
<td>Publishing and Consuming REST-Based XML Web Services</td>
</tr>
<tr>
<td>28.7.1</td>
<td>Creating a REST-Based XML Web Service</td>
</tr>
<tr>
<td>28.7.2</td>
<td>Consuming a REST-Based XML Web Service</td>
</tr>
<tr>
<td>28.8</td>
<td>Publishing and Consuming REST-Based JSON Web Services</td>
</tr>
<tr>
<td>28.8.1</td>
<td>Creating a REST-Based JSON Web Service</td>
</tr>
<tr>
<td>28.8.2</td>
<td>Consuming a REST-Based JSON Web Service</td>
</tr>
</tbody>
</table>
Contents

28.9 Session Tracking in a SOAP Web Service 987
28.9.1 Creating a Blackjack Web Service 988
28.9.2 Consuming the Blackjack Web Service 991
28.10 Consuming a Database-Driven SOAP Web Service 1002
28.10.1 Creating the Reservation Database 1003
28.10.2 Creating a Web Application to Interact with the Reservation Service 1006
28.11 Equation Generator: Returning User-Defined Types 1009
28.11.1 Creating the EquationGeneratorXML Web Service 1012
28.11.2 Consuming the EquationGeneratorXML Web Service 1013
28.11.3 Creating the EquationGeneratorJSON Web Service 1017
28.11.4 Consuming the EquationGeneratorJSON Web Service 1017
28.12 Wrap-Up 1020

A Operator Precedence Chart 1022

B ASCII Character Set 1024

C Keywords and Reserved Words 1025

D Primitive Types 1026

E Using the Java API Documentation 1027
E.1 Introduction 1027
E.2 Navigating the Java API 1028

F Using the Debugger 1036
F.1 Introduction 1037
F.2 Breakpoints and the run, stop, cont and print Commands 1041
F.3 The print and set Commands 1043
F.4 Controlling Execution Using the step, step up and next Commands 1046
F.5 The watch Command 1049
F.6 The clear Command 1049
F.7 Wrap-Up 1051

G Formatted Output 1052
G.1 Introduction 1053
G.2 Streams 1053
G.3 Formatting Output with printf 1053
This page intentionally left blank
Live in fragments no longer, only connect.
—Edgar Morgan Foster

Welcome to Java and *Java for Programmers, Second 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”—concepts are presented in the context of complete working programs, rather than in code snippets. Each complete code example is accompanied by live sample executions. All the source code is available at

www.deitel.com/books/javafp2/

As you read the book, if you have questions, send an e-mail to deitel@deitel.com; we’ll respond promptly. For updates on this book, visit the website shown above, follow us on Facebook (www.facebook.com/DeitelFan) and Twitter (@deitel), and subscribe to the *Deitel® Buzz Online* newsletter (www.deitel.com/newsletter/subscribe.html).

**Features**

Here are the key features of *Java for Programmers, 2/e*:

**Java Standard Edition (SE) 7**

- **Easy to use as a Java SE 6 or Java SE 7 book.** We cover the new Java SE 7 features in modular sections. Here’s some of the new functionality: Strings in `switch` statements, the `try-with-resources` statement for managing `AutoClosable` objects, multi-catch for defining a single exception handler to replace multiple exception handlers that perform the same task and inferring the types of generic objects from the variable they’re assigned to by using the `<>` notation. We also overview the new concurrency API features.

- **Java SE 7’s `AutoClosable` versions of `Connection`, `Statement` and `ResultSet`**. With the source code for Chapter 25, Accessing Databases with JDBC, we provide a version of the chapter’s first example that’s implemented using Java SE 7’s `AutoClosable` versions of `Connection`, `Statement` and `ResultSet`. `AutoClosable` objects reduce the likelihood of resource leaks when you use them with Java SE 7’s `try-with-resources` statement, which automatically closes the `AutoClosable` objects allocated in the parentheses following the `try` keyword.

**Object Technology**

- **Object-oriented programming and design.** We review the basic concepts and terminology of object technology in Chapter 1. Readers develop their first customized classes and objects in Chapter 3.
• **Exception handling.** We integrate basic exception handling early in the book and cover it in detail in Chapter 11, Exception Handling: A Deeper Look.

• **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.

• **OO case studies.** The early classes and objects presentation features Time, Employee and GradeBook class case studies that weave their way through multiple sections and chapters, gradually introducing deeper OO concepts.

• **Case Study: Using the UML to Develop an Object-Oriented Design and Java Implementation of an ATM.** The UML™ (Unified Modeling Language™) is the industry-standard graphical language for modeling object-oriented systems. Chapters 12–13 include a case study on object-oriented design using the UML. 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 complete Java implementation. Readers often report having a “light-bulb moment”—the case study helps them “tie it all together” and really understand object orientation in Java.

• **Reordered generics presentation.** We begin with generic class ArrayList in Chapter 7. Because you’ll understand basic generics concepts early in the book, our later data structures discussions provide a deeper treatment of generic collections—showing how to use the built-in collections of the Java API. We then show how to implement generic methods and classes.

**Database and Web Development**

• **JDBC 4.** Chapter 25, Accessing Databases with JDBC, covers JDBC 4 and uses the Java DB/Apache Derby and MySQL database management systems. The chapter features an OO case study on developing a database-driven address book that demonstrates prepared statements and JDBC 4’s automatic driver discovery.

• **Java Server Faces (JSF) 2.0.** Chapters 26–27 have been updated with JavaServer Faces (JSF) 2.0 technology, which greatly simplifies building JSF web applications. Chapter 26 includes examples on building web application GUIs, validating forms and session tracking. Chapter 27 discusses data-driven and Ajax-enabled JSF applications. The chapter features a database-driven multitier web address book that allows users to add and search for contacts.

• **Web services.** Chapter 28, Web Services, demonstrates creating and consuming SOAP- and REST-based web services. Case studies include developing blackjack and airline reservation web services.

• **Java Web Start and the Java Network Launch Protocol (JNLP).** We introduce Java Web Start and JNLP, which enable applets and applications to be launched via a web browser. Users can install locally for later execution. Programs can also request the user’s permission to access local system resources such as files—en-
Teaching Approach

Java for Programmers, 2/e, contains hundreds of complete working examples. We stress program clarity and concentrate on building well-engineered software.

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

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

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;).

abling you to develop more robust applets and applications that execute safely using Java’s sandbox security model, which applies to downloaded code.

Multithreading

- **Multithreading.** We completely reworked Chapter 23, Multithreading [special thanks to the guidance of Brian Goetz and Joseph Bowbeer—two of the co-authors of *Java Concurrency in Practice*, Addison-Wesley, 2006].

- **SwingWorker class.** We use class SwingWorker to create multithreaded user interfaces.

GUI and Graphics

- **GUI and graphics presentation.** Chapters 14, 15 and 22, and Appendix H present Java GUI and Graphics programming.

- **GridLayout layout manager.** We discuss the GridLayout layout manager in the context of the GUI design tool in the NetBeans IDE.

- **JTable sorting and filtering capabilities.** Chapter 25 uses these capabilities to sort the data in a JTable and filter it by regular expressions.

Other Features

- **Android.** Because of the tremendous interest in Android-based smartphones and tablets, we’ve included a three-chapter introduction to Android app development online at www.deitel.com/books/javafp. These chapters are from our new Deitel Developer Series book *Android for Programmers: An App-Driven Approach*. After you learn Java, you’ll find it straightforward to develop and run Android apps on the free Android emulator that you can download from developer.android.com.

- **Software engineering community concepts.** We discuss agile software development, refactoring, design patterns, LAMP, SaaS (Software as a Service), PaaS (Platform as a Service), cloud computing, open-source software and more.
Web Access. All of the source-code examples can be downloaded from:

www.deitel.com/books/javafp2
www.pearsonhighered.com/deitel

Objectives. The chapter 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 eight 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 the same errors.

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
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
These observations help you design attractive, user-friendly graphical user interfaces that conform to industry norms.

Thousands of Index Entries. We’ve included a comprehensive index, which is especially useful when you use the book as a reference.

Software Used in Java for Programmers, 2/e
All the software you’ll need for this book is available free for download from the web. See the Before You Begin section that follows the Preface for links to each download.
We wrote most of the examples in *Java for Programmers, 2/e*, using the free Java Standard Edition Development Kit (JDK) 6. For the Java SE 7 modules, we used the OpenJDK’s early access version of JDK 7 (download.java.net/jdk7/). In Chapters 26–28, we also used the Netbeans IDE, and in Chapter 25, we used MySQL and MySQL Connector/J. You can find additional resources and software downloads in our Java Resource Centers at:

www.deitel.com/ResourceCenters.html

**Discounts on Deitel Developer Series Books**

If you’d like to receive information on professional Deitel Developer Series titles, including *Android for Programmers: An App-Driven Approach*, please register your copy of *Java for Programmers, 2/e* at informit.com/register. You’ll receive information on how to purchase *Android for Programmers* at a discount.


Our *Java Fundamentals: Parts I, II and III, Second Edition* LiveLessons video training product shows you what you need to know to start building robust, powerful software with Java. It includes 20+ hours of expert training synchronized with *Java for Programmers, 2/e*. Check out our growing list of LiveLessons video products:

- Java Fundamentals I and II
- C# 2010 Fundamentals I, II, and III
- C# 2008 Fundamentals I and II
- C++ Fundamentals I and II
- iPhone App-Development Fundamentals I and II
- JavaScript Fundamentals I and II
- Visual Basic 2010 Fundamentals I and II

**Coming Soon**

- Java Fundamentals I, II and III, Second Edition
- C Fundamentals I and II
- Android App Development Fundamentals I and II
- iPhone and iPad App-Development Fundamentals I and II, Second Edition

For additional information about Deitel LiveLessons video products, visit:

www.deitel.com/livelessons

**Acknowledgments**

We’d like to thank Abbey Deitel and Barbara Deitel for long hours devoted to this project. Barbara devoted long hours to Internet research to support our writing efforts. Abbey wrote the new engaging Chapter 1 and the new cover copy. We’re fortunate to have worked on this project with the dedicated team of publishing professionals at Pearson. We appreciate
the guidance, savvy and energy of Mark Taub, Editor-in-Chief of Computer Science. John Fuller managed the book’s production. Sandra Schroeder did the cover design.

Reviewers
We wish to acknowledge the efforts of the reviewers who contributed to the recent editions of this content. They scrutinized the text and the programs and provided countless suggestions for improving the presentation: Lance Andersen (Oracle), 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), Huiwei Guan (Northshore Community College), 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 (Java Champion, Consultant), Manjeet Rege, Ph.D. (Rochester Institute of Technology), Manfred Riem (Java Champion, Consultant, Robert Half), Simon Ritter (Oracle), 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).

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 for Programmers, 2/e. Good luck!

Paul and Harvey Deitel

About the Authors
Paul J. Deitel, CEO and Chief Technical Officer of Deitel & Associates, Inc., is a graduate of MIT, where he studied Information Technology. He holds the Sun (now Oracle) Certified Java Programmer and Certified Java Developer certifications, and is an Oracle Java Champion. Through Deitel & Associates, Inc., he has delivered Java, C#, Visual Basic, C++, C and Internet programming courses to industry clients, including Cisco, IBM, Sun Microsystems, Dell, Siemens, Lucent Technologies, Fidelity, NASA at the Kennedy Space Center, the National Severe Storm Laboratory, White Sands Missile Range, Rogue Wave Software, Boeing, SunGard Higher Education, Stratus, Cambridge Technology Partners, One Wave, Hyperion Software, Adra Systems, Entergy, CableData Systems, 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 authors.

Dr. Harvey M. Deitel, Chairman and Chief Strategy Officer of Deitel & Associates, Inc., has 50 years of experience in the computer field. Dr. Deitel earned B.S. and M.S. degrees from MIT and a Ph.D. from Boston University. He has extensive industry and academic experience, including earning tenure and serving as the Chairman of the Computer Science Department at Boston College before founding Deitel & Associates, Inc.,
with his son, Paul J. Deitel. He and Paul are the co-authors of dozens of books and multimedia packages and they are writing many more. With translations published in Japanese, German, Russian, Chinese, Spanish, Korean, French, Polish, Italian, Portuguese, Greek, Urdu and Turkish, the Deitels’ texts have earned international recognition. Dr. Deitel has delivered hundreds of professional seminars to major corporations, academic institutions, government organizations and the military.

About Deitel & Associates, Inc.

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

Through its 35-year publishing partnership with Prentice Hall/Pearson, Deitel & Associates, Inc., publishes leading-edge programming professional books, college textbooks, and LiveLessons DVD- and web-based 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:

www.deitel.com/training/

subscribe to the free Deitel® Buzz Online e-mail newsletter at:

www.deitel.com/newsletter/subscribe.html

and follow the authors on Facebook

www.facebook.com/DeitelFan

and Twitter

@deitel

To request a proposal for on-site, instructor-led training at your company or organization, e-mail

deitel@deitel.com

Individuals wishing to purchase Deitel books and LiveLessons DVD training courses 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 www.pearsoned.com/professional/index.htm.
This page intentionally left blank
Before You Begin

This section contains information you should review before using this book and instructions to ensure that your computer is set up properly for use with this book. We’ll post updates (if any) to the Before You Begin section on the book’s website:

www.deitel.com/books/javafp2/

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.

Java SE Software Development Kit (JDK) 6 and 7

We wrote most of the examples in Java for Programmers, 2/e, using the free Java Standard Edition Development Kit (JDK) 6, which is available from:

www.oracle.com/technetwork/java/javase/downloads/index.html

For the Java SE 7 modules, we used the OpenJDK’s early access version of JDK 7, which is available from:

dl.c.sun.com/edgesuite.net/jdk7/binaries-/index.html

Java DB, MySQL and MySQL Connector/J

In Chapter 25, we use the Java DB and MySQL Community Edition database management systems. Java DB is part of the JDK installation. At the time of this writing, the JDK’s 64-bit installer was not properly installing Java DB. If you are using the 64-bit version of Java, you may need to install Java DB separately. You can download Java DB from:


At the time of this writing, the latest release of MySQL Community Edition was 5.5.8. To install MySQL Community Edition on Windows, Linux or Mac OS X, see the installation overview for your platform at:

- Linux: dev.mysql.com/doc/refman/5.5/en/linux-installation-rpm.html
Carefully follow the instructions for downloading and installing the software on your platform. The downloads are available from:

```
dev.mysql.com/downloads/mysql/
```

You also need to install MySQL Connector/J (the J stands for Java), which allows programs to use JDBC to interact with MySQL. MySQL Connector/J can be downloaded from:

```
dev.mysql.com/downloads/connector/j/
```

At the time of this writing, the current generally available release of MySQL Connector/J is 5.1.14. The documentation for Connector/J is located at:

```
dev.mysql.com/doc/refman/5.5/en/connector-j.html
```

To install MySQL Connector/J, carefully follow the installation instructions at:

```
dev.mysql.com/doc/refman/5.5/en/connector-j-installing.html
```

We do not recommend modifying your system’s CLASSPATH environment variable, which is discussed in the installation instructions. Instead, we’ll show you how use MySQL Connector/J by specifying it as a command-line option when you execute your applications.

## Obtaining the Code Examples

The examples for *Java for Programmers, 2/e* are available for download at:

```
www.deitel.com/books/javafp2/
```

If you’re not already registered at our website, go to www.deitel.com and click the Register link below our logo in the upper-left corner of the page. Fill in your information. There’s no charge to register, and we do not share your information with anyone. We send you only account-management e-mails unless you register separately for our free Deitel® Buzz Online e-mail newsletter at www.deitel.com/newsletter/subscribe.html. After registering for the site, you’ll receive a confirmation e-mail with your verification code. Click the link in the confirmation e-mail to complete your registration. Configure your e-mail client to allow e-mails from deitel.com to ensure that the confirmation email is not filtered as junk mail.

Next, go to www.deitel.com and sign in using the Login link below our logo in the upper-left corner of the page. Go to www.deitel.com/books/javafp2/. You’ll find the link to download the examples under the heading *Download Code Examples and Other Premium Content for Registered Users*. Write down the location where you choose to save the ZIP file on your computer. We assume the examples are located at C:\Examples on your computer.

## 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.
If you do not set the `PATH` variable correctly, 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.

### Setting the CLASSPATH Environment Variable

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

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

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

```
;
```

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

### Java’s Nimbus Look-and-Feel

Java comes bundled with an elegant, cross-platform look-and-feel known as Nimbus. For programs with graphical user interfaces, we’ve configured our systems 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 installation folders visit java.sun.com/javase/6/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.]
This page intentionally left blank
Object-Oriented Programming: Polymorphism

Objectives
In this chapter you’ll learn:

- The concept of polymorphism.
- To use overridden methods to effect polymorphism.
- To distinguish between abstract and concrete classes.
- To declare abstract methods to create abstract classes.
- How polymorphism makes systems extensible and maintainable.
- To determine an object’s type at execution time.
- To declare and implement interfaces.

One Ring to rule them all,
One Ring to find them,
One Ring to bring them all
and in the darkness bind
them.
—John Ronald Reuel Tolkien

General propositions do not decide concrete cases.
—Oliver Wendell Holmes

A philosopher of imposing stature doesn’t think in a vacuum. Even his most abstract ideas are, to some extent, conditioned by what is or is not known in the time when he lives.
—Alfred North Whitehead

Why art thou cast down, O my soul?
—Psalms 42:5
10.1 Introduction

We continue our study of object-oriented programming by explaining and demonstrating **polymorphism** with inheritance hierarchies. Polymorphism enables you to “program in the general” rather than “program in the specific.” In particular, polymorphism enables you to write programs that process objects that share the same superclass (either directly or indirectly) as if they’re all objects of the superclass; this can simplify programming.

Consider the following example of polymorphism. Suppose we create a program that simulates the movement of several types of animals for a biological study. Classes *Fish*, *Frog* and *Bird* represent the types of animals under investigation. Imagine that each class extends superclass *Animal*, which contains a method *move* and maintains an animal’s current location as $x$-$y$ coordinates. Each subclass implements method *move*. Our program maintains an *Animal* array containing references to objects of the various *Animal* subclasses. To simulate the animals’ movements, the program sends each object the *same* message once per second—namely, *move*. Each specific type of *Animal* responds to a *move* message in its own way—a *Fish* might swim three feet, a *Frog* might jump five feet and a *Bird* might fly ten feet. Each object knows how to modify its $x$-$y$ coordinates appropriately for its *specific* type of movement. Relying on each object to know how to “do the right thing” (i.e., do what is appropriate for that type of object) in response to the same method call is the key concept of polymorphism. The same message (in this case, *move*) sent to a variety of objects has “many forms” of results—hence the term polymorphism.

**Implementing for Extensibility**

With polymorphism, we can design and implement systems that are easily extensible—new classes can be added with little or no modification to the general portions of the program, as long as the new classes are part of the inheritance hierarchy that the program processes generically. The only parts of a program that must be altered are those that require direct knowledge of the new classes that we add to the hierarchy. For example, if we extend
class Animal to create class Tortoise (which might respond to a move message by crawling one inch), we need to write only the Tortoise class and the part of the simulation that instantiates a Tortoise object. The portions of the simulation that tell each Animal to move generically can remain the same.

**Chapter Overview**

First, we discuss common examples of polymorphism. We then provide a simple example demonstrating polymorphic behavior. We use superclass references to manipulate both superclass objects and subclass objects polymorphically.

We then present a case study that revisits the employee hierarchy of Section 9.4.5. We develop a simple payroll application that polymorphically calculates the weekly pay of several different types of employees using each employee’s earnings method. Though the earnings of each type of employee are calculated in a specific way, polymorphism allows us to process the employees “in the general.” In the case study, we enlarge the hierarchy to include two new classes—SalariedEmployee (for people paid a fixed weekly salary) and HourlyEmployee (for people paid an hourly salary and “time-and-a-half” for overtime). We declare a common set of functionality for all the classes in the updated hierarchy in an “abstract” class, Employee, from which “concrete” classes SalariedEmployee, HourlyEmployee and CommissionEmployee inherit directly and “concrete” class BasePlusCommissionEmployee inherits indirectly. As you’ll soon see, when we invoke each employee’s earnings method off a superclass Employee reference, the correct earnings subclass calculation is performed, due to Java’s polymorphic capabilities.

**Programming in the Specific**

Occasionally, when performing polymorphic processing, we need to program “in the specific.” Our Employee case study demonstrates that a program can determine the type of an object at execution time and act on that object accordingly. In the case study, we’ve decided that BasePlusCommissionEmployees should receive 10% raises on their base salaries. So, we use these capabilities to determine whether a particular employee object is a BasePlusCommissionEmployee. If so, we increase that employee’s base salary by 10%.

**Interfaces**

The chapter continues with an introduction to Java interfaces. An interface describes a set of methods that can be called on an object, but does not provide concrete implementations for all the methods. You can declare classes that implement (i.e., provide concrete implementations for the methods of) one or more interfaces. Each interface method must be declared in all the classes that explicitly implement the interface. Once a class implements an interface, all objects of that class have an is-a relationship with the interface type, and all objects of the class are guaranteed to provide the functionality described by the interface. This is true of all subclasses of that class as well.

Interfaces are particularly useful for assigning common functionality to possibly unrelated classes. This allows objects of unrelated classes to be processed polymorphically—objects of classes that implement the same interface can respond to all of the interface method calls. To demonstrate creating and using interfaces, we modify our payroll application to create a general accounts payable application that can calculate payments due for company employees and invoice amounts to be billed for purchased goods. As you’ll see, interfaces enable polymorphic capabilities similar to those possible with inheritance.
10.2 Polymorphism Examples

We now consider several additional examples of polymorphism.

Quadrilaterals
If class Rectangle is derived from class Quadrilateral, then a Rectangle object is a more specific version of a Quadrilateral. Any operation (e.g., calculating the perimeter or the area) that can be performed on a Quadrilateral can also be performed on a Rectangle. These operations can also be performed on other Quadrilaterals, such as Squares, Parallelograms and Trapezoids. The polymorphism occurs when a program invokes a method through a superclass Quadrilateral variable—at execution time, the correct subclass version of the method is called, based on the type of the reference stored in the superclass variable. You’ll see a simple code example that illustrates this process in Section 10.3.

Space Objects in a Video Game
Suppose we design a video game that manipulates objects of classes Martian, Venusian, Plutonian, SpaceShip and LaserBeam. Imagine that each class inherits from the superclass SpaceObject, which contains method draw. Each subclass implements this method. A screen manager maintains a collection (e.g., a SpaceObject array) of references to objects of the various classes. To refresh the screen, the screen manager periodically sends each object the same message—namely, draw. However, each object responds its own way, based on its class. For example, a Martian object might draw itself in red with green eyes and the appropriate number of antennae. A SpaceShip object might draw itself as a bright silver flying saucer. A LaserBeam object might draw itself as a bright red beam across the screen. Again, the same message (in this case, draw) sent to a variety of objects has “many forms” of results.

A screen manager might use polymorphism to facilitate adding new classes to a system with minimal modifications to the system’s code. Suppose that we want to add Mercurian objects to our video game. To do so, we’d build a class Mercurian that extends SpaceObject and provides its own draw method implementation. When Mercurian objects appear in the SpaceObject collection, the screen manager code invokes method draw, exactly as it does for every other object in the collection, regardless of its type. So the new Mercurian objects simply “plug right in” without any modification of the screen manager code by the programmer. Thus, without modifying the system (other than to build new classes and modify the code that creates new objects), you can use polymorphism to conveniently include additional types that were not envisioned when the system was created.

Software Engineering Observation 10.1
Polymorphism enables you to deal in generalities and let the execution-time environment handle the specifics. You can command objects to behave in manners appropriate to those objects, without knowing their types (as long as the objects belong to the same inheritance hierarchy).

Software Engineering Observation 10.2
Polymorphism promotes extensibility: Software that invokes polymorphic behavior is independent of the object types to which messages are sent. New object types that can respond to existing method calls can be incorporated into a system without modifying the base system. Only client code that instantiates new objects must be modified to accommodate new types.
10.3 Demonstrating Polymorphic Behavior

Section 9.4 created a class hierarchy, in which class BasePlusCommissionEmployee inherited from CommissionEmployee. The examples in that section manipulated CommissionEmployee and BasePlusCommissionEmployee objects by using references to them to invoke their methods—we aimed superclass variables at superclass objects and subclass variables at subclass objects. These assignments are natural and straightforward—superclass variables are intended to refer to superclass objects, and subclass variables are intended to refer to subclass objects. However, as you’ll soon see, other assignments are possible.

In the next example, we aim a superclass reference at a subclass object. We then show how invoking a method on a subclass object via a superclass reference invokes the subclass functionality—the type of the referenced object, not the type of the variable, determines which method is called. This example demonstrates that an object of a subclass can be treated as an object of its superclass, enabling various interesting manipulations. A program can create an array of superclass variables that refer to objects of many subclass types. This is allowed because each subclass object is an object of its superclass. For instance, we can assign the reference of a BasePlusCommissionEmployee object to a superclass CommissionEmployee variable, because a BasePlusCommissionEmployee is a CommissionEmployee—we can treat a BasePlusCommissionEmployee as a CommissionEmployee.

As you’ll learn later in the chapter, you cannot treat a superclass object as a subclass object, because a superclass object is not an object of any of its subclasses. For example, we cannot assign the reference of a CommissionEmployee object to a subclass BasePlusCommissionEmployee variable, because a CommissionEmployee is not a BasePlusCommissionEmployee—a CommissionEmployee does not have a baseSalary instance variable and does not have methods setBaseSalary and getBaseSalary. The is-a relationship applies only up the hierarchy from a subclass to its direct (and indirect) superclasses, and not vice versa (i.e., not down the hierarchy from a superclass to its subclasses).

The Java compiler does allow the assignment of a superclass reference to a subclass variable if we explicitly cast the superclass reference to the subclass type—a technique we discuss in Section 10.5. Why would we ever want to perform such an assignment? A superclass reference can be used to invoke only the methods declared in the superclass—attempting to invoke subclass-only methods through a superclass reference results in compilation errors. If a program needs to perform a subclass-specific operation on a subclass object referenced by a superclass variable, the program must first cast the superclass reference to a subclass reference through a technique known as downcasting. This enables the program to invoke subclass methods that are not in the superclass. We show a downcasting example in Section 10.5.

The example in Fig. 10.1 demonstrates three ways to use superclass and subclass variables to store references to superclass and subclass objects. The first two are straightforward—as in Section 9.4, we assign a superclass reference to a superclass variable, and a subclass reference to a subclass variable. Then we demonstrate the relationship between subclasses and superclasses (i.e., the is-a relationship) by assigning a subclass reference to a superclass variable. This program uses classes CommissionEmployee and BasePlusCommissionEmployee from Fig. 9.10 and Fig. 9.11, respectively.

In Fig. 10.1, lines 10–11 create a CommissionEmployee object and assign its reference to a CommissionEmployee variable. Lines 14–16 create a BasePlusCommissionEmployee object and assign its reference to a BasePlusCommissionEmployee variable. These assign-
```java
public class PolymorphismTest {
    public static void main(String[] args) {
        // assign superclass reference to superclass variable
        CommissionEmployee commissionEmployee = new CommissionEmployee("Sue", "Jones", "222-22-2222", 10000, .06);
        // assign subclass reference to subclass variable
        BasePlusCommissionEmployee basePlusCommissionEmployee =
            new BasePlusCommissionEmployee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300);

        // invoke toString on superclass object using superclass variable
        System.out.printf("%s %s:

%s

", "Call CommissionEmployee's toString with superclass reference",
          "to superclass object", commissionEmployee.toString());

        // invoke toString on subclass object using subclass variable
        System.out.printf("%s %s:

%s

", "Call BasePlusCommissionEmployee's toString with subclass",
          "reference to subclass object",
          basePlusCommissionEmployee.toString());

        // invoke toString on subclass object using superclass variable
        CommissionEmployee commissionEmployee2 = basePlusCommissionEmployee;
        System.out.printf("%s %s:

%s

", "Call BasePlusCommissionEmployee's toString with superclass",
          "reference to subclass object", commissionEmployee2.toString());
    } // end main
} // end class PolymorphismTest
```

Fig. 10.1 | Assigning superclass and subclass references to superclass and subclass variables.
(Part 1 of 2.)
ments are natural—for example, a CommissionEmployee variable’s primary purpose is to hold a reference to a CommissionEmployee object. Lines 19–21 use commissionEmployee to invoke toString explicitly. Because commissionEmployee refers to a CommissionEmployee object, superclass CommissionEmployee’s version of toString is called. Similarly, lines 24–27 use basePlusCommissionEmployee to invoke toString explicitly on the BasePlusCommissionEmployee object. This invokes subclass BasePlusCommissionEmployee’s version of toString.

Lines 30–31 then assign the reference of subclass object basePlusCommissionEmployee to a superclass CommissionEmployee variable, which lines 32–34 use to invoke method toString. When a superclass variable contains a reference to a subclass object, and that reference is used to call a method, the subclass version of the method is called. Hence, commissionEmployee2.toString() in line 34 actually calls class BasePlusCommissionEmployee’s toString method. The Java compiler allows this “crossover” because an object of a subclass is an object of its superclass (but not vice versa). When the compiler encounters a method call made through a variable, the compiler determines if the method can be called by checking the variable’s class type. If that class contains the proper method declaration (or inherits one), the call is compiled. At execution time, the type of the object to which the variable refers determines the actual method to use. This process, called \textit{dynamic binding}, is discussed in detail in Section 10.5.

\section*{10.4 Abstract Classes and Methods}

When we think of a class, we assume that programs will create objects of that type. Sometimes it’s useful to declare classes—called abstract classes—for which you never intend to create objects. Because they’re used only as superclasses in inheritance hierarchies, we refer to them as \textit{abstract superclasses}. These classes cannot be used to instantiate objects, because, as we’ll soon see, abstract classes are \textit{incomplete}. Subclasses must declare the “missing pieces” to become “concrete” classes, from which you can instantiate objects. Otherwise, these subclasses, too, will be abstract. We demonstrate abstract classes in Section 10.5.

\textit{Purpose of Abstract Classes}

An abstract class’s purpose is to provide an appropriate superclass from which other classes can inherit and thus share a common design. In the Shape hierarchy of Fig. 9.3, for example, subclasses inherit the notion of what it means to be a Shape—perhaps common attributes such as location, color and borderThickness, and behaviors such as draw, move, resize and changeColor. Classes that can be used to instantiate objects are called \textit{concrete}.

\begin{figure}[h]
\centering
\begin{tabular}{l}
Call BasePlusCommissionEmployee's toString with superclass reference to subclass object:
\\
base-salaried commission employee: Bob Lewis \\
social security number: 333-33-3333 \\
gross sales: 5000.00 \\
commission rate: 0.04 \\
base salary: 300.00
\end{tabular}
\caption{Assigning superclass and subclass references to superclass and subclass variables.}
\end{figure}

\textit{Fig. 10.1} | Assigning superclass and subclass references to superclass and subclass variables. (Part 2 of 2.)
Abstract classes. Such classes provide implementations of every method they declare (some of the implementations can be inherited). For example, we could derive concrete classes Circle, Square and Triangle from abstract superclass TwoDimensionalShape. Similarly, we could derive concrete classes Sphere, Cube and Tetrahedron from abstract superclass ThreeDimensionalShape. Abstract superclasses are too general to create real objects—they specify only what is common among subclasses. We need to be more specific before we can create objects. For example, if you send the draw message to abstract class TwoDimensionalShape, the class knows that two-dimensional shapes should be drawable, but it does not know what specific shape to draw, so it cannot implement a real draw method. Concrete classes provide the specifics that make it reasonable to instantiate objects.

Not all hierarchies contain abstract classes. However, you’ll often write client code that uses only abstract superclass types to reduce the client code’s dependencies on a range of subclass types. For example, you can write a method with a parameter of an abstract superclass type. When called, such a method can receive an object of any concrete class that directly or indirectly extends the superclass specified as the parameter’s type.

Abstract classes sometimes constitute several levels of a hierarchy. For example, the Shape hierarchy of Fig. 9.3 begins with abstract class Shape. On the next level of the hierarchy are abstract classes TwoDimensionalShape and ThreeDimensionalShape. The next level of the hierarchy declares concrete classes for TwoDimensionalShapes (Circle, Square and Triangle) and for ThreeDimensionalShapes (Sphere, Cube and Tetrahedron).

Declaring an Abstract Class and Abstract Methods
You make a class abstract by declaring it with keyword abstract. An abstract class normally contains one or more abstract methods. An abstract method is one with keyword abstract in its declaration, as in

```java
public abstract void draw(); // abstract method
```

Abstract methods do not provide implementations. A class that contains any abstract methods must be explicitly declared abstract even if that class contains some concrete (nonabstract) methods. Each concrete subclass of an abstract superclass also must provide concrete implementations of each of the superclass’s abstract methods. Constructors and static methods cannot be declared abstract. Constructors are not inherited, so an abstract constructor could never be implemented. Though non-private static methods are inherited, they cannot be overridden. Since abstract methods are meant to be overridden so that they can process objects based on their types, it would not make sense to declare a static method as abstract.

Software Engineering Observation 10.3
An abstract class declares common attributes and behaviors (both abstract and concrete) of the various classes in a class hierarchy. An abstract class typically contains one or more abstract methods that subclasses must override if they are to be concrete. The instance variables and concrete methods of an abstract class are subject to the normal rules of inheritance.

Common Programming Error 10.1
Attempting to instantiate an object of an abstract class is a compilation error.
Although we cannot instantiate objects of abstract superclasses, you’ll soon see that we can use abstract superclasses to declare variables that can hold references to objects of any concrete class derived from those abstract superclasses. Programs typically use such variables to manipulate subclass objects polymorphically. You also can use abstract superclass names to invoke static methods declared in those abstract superclasses.

Consider another application of polymorphism. A drawing program needs to display many shapes, including types of new shapes that you’ll add to the system after writing the drawing program. The drawing program might need to display shapes, such as Circles, Triangles, Rectangles or others, that derive from abstract class Shape. The drawing program uses Shape variables to manage the objects that are displayed. To draw any object in this inheritance hierarchy, the drawing program uses a superclass Shape variable containing a reference to the subclass object to invoke the object’s draw method. This method is declared abstract in superclass Shape, so each concrete subclass must implement method draw in a manner specific to that shape—each object in the Shape inheritance hierarchy knows how to draw itself. The drawing program does not have to worry about the type of each object or whether the program has ever encountered objects of that type.

Layered Software Systems
Polymorphism is particularly effective for implementing so-called layered software systems. In operating systems, for example, each type of physical device could operate quite differently from the others. Even so, commands to read or write data from and to devices may have a certain uniformity. For each device, the operating system uses a piece of software called a device driver to control all communication between the system and the device. The write message sent to a device-driver object needs to be interpreted specifically in the context of that driver and how it manipulates devices of a specific type. However, the write call itself really is no different from the write to any other device in the system—place some number of bytes from memory onto that device. An object-oriented operating system might use an abstract superclass to provide an “interface” appropriate for all device drivers. Then, through inheritance from that abstract superclass, subclasses are formed that all behave similarly. The device-driver methods are declared as abstract methods in the abstract superclass. The implementations of these abstract methods are provided in the concrete subclasses that correspond to the specific types of device drivers. New devices are always being developed, often long after the operating system has been released. When you buy a new device, it comes with a device driver provided by the device vendor. The device is immediately operational after you connect it to your computer and install the driver. This is another elegant example of how polymorphism makes systems extensible.

10.5 Case Study: Payroll System Using Polymorphism
This section reexamines the hierarchy that we explored throughout Section 9.4. Now we use an abstract method and polymorphism to perform payroll calculations based on an enhanced employee inheritance hierarchy that meets the following requirements:
A company pays its employees on a weekly basis. The employees are of four types: Salaried employees are paid a fixed weekly salary regardless of the number of hours worked, hourly employees are paid by the hour and receive overtime pay (i.e., 1.5 times their hourly salary rate) for all hours worked in excess of 40 hours, commission employees are paid a percentage of their sales and base-salaried commission employees receive a base salary plus a percentage of their sales. For the current pay period, the company has decided to reward salaried-commission employees by adding 10% to their base salaries. The company wants to write an application that performs its payroll calculations polymorphically.

We use abstract class Employee to represent the general concept of an employee. The classes that extend Employee are SalariedEmployee, CommissionEmployee and HourlyEmployee. Class BasePlusCommissionEmployee—which extends CommissionEmployee—represents the last employee type. The UML class diagram in Fig. 10.2 shows the inheritance hierarchy for our polymorphic employee-payroll application. Abstract class name Employee is italicized—a convention of the UML.

![Fig. 10.2 Employee hierarchy UML class diagram.](image)

Abstract superclass Employee declares the “interface” to the hierarchy—that is, the set of methods that a program can invoke on all Employee objects. We use the term “interface” here in a general sense to refer to the various ways programs can communicate with objects of any Employee subclass. Be careful not to confuse the general notion of an “interface” with the formal notion of a Java interface, the subject of Section 10.7. Each employee, regardless of the way his or her earnings are calculated, has a first name, a last name and a social security number, so private instance variables firstName, lastName and socialSecurityNumber appear in abstract superclass Employee.

The following sections implement the Employee class hierarchy of Fig. 10.2. The first section implements abstract superclass Employee. The next four sections each implement one of the concrete classes. The last section implements a test program that builds objects of all these classes and processes those objects polymorphically.

### 10.5.1 Abstract Superclass Employee

Class Employee (Fig. 10.4) provides methods earnings and toString, in addition to the get and set methods that manipulate Employee’s instance variables. An earnings method certainly applies generically to all employees. But each earnings calculation depends on the employee’s class. So we declare earnings as abstract in superclass Employee because a de-
fault implementation does not make sense for that method—there isn’t enough information to determine what amount earnings should return. Each subclass overrides earnings with an appropriate implementation. To calculate an employee’s earnings, the program assigns to a superclass Employee variable a reference to the employee’s object, then invokes the earnings method on that variable. We maintain an array of Employee variables, each holding a reference to an Employee object. (Of course, there cannot be Employee objects, because Employee is an abstract class. Because of inheritance, however, all objects of all subclasses of Employee may nevertheless be thought of as Employee objects.) The program will iterate through the array and call method earnings for each Employee object. Java processes these method calls polymorphically. Declaring earnings as an abstract method in Employee enables the calls to earnings through Employee variables to compile and forces every direct concrete subclass of Employee to override earnings.

Method toString in class Employee returns a String containing the first name, last name and social security number of the employee. As we’ll see, each subclass of Employee overrides method toString to create a String representation of an object of that class that contains the employee’s type (e.g., “salaried employee:”) followed by the rest of the employee’s information.

The diagram in Fig. 10.3 shows each of the five classes in the hierarchy down the left side and methods earnings and toString across the top. For each class, the diagram

<table>
<thead>
<tr>
<th>earnings</th>
<th>toString</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee</td>
<td>abstract</td>
</tr>
<tr>
<td>Salaried-Employee</td>
<td>weeklySalary</td>
</tr>
<tr>
<td>Hourly-Employee</td>
<td>if (hours &lt;= 40) wage * hours else if (hours &gt; 40) { 40 * wage + (hours - 40) * wage * 1.5 }</td>
</tr>
<tr>
<td>Commission-Employee</td>
<td>commissionRate * grossSales</td>
</tr>
<tr>
<td>BasePlus-Commission-Employee</td>
<td>(commissionRate * grossSales) + baseSalary</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th>String representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salaried employee: firstName lastName social security number: SSN</td>
</tr>
<tr>
<td>Hourly employee: firstName lastName social security number: SSN hourly wage: wage; hours worked: hours</td>
</tr>
<tr>
<td>Commission employee: firstName lastName social security number: SSN gross sales: grossSales; commission rate: commissionRate</td>
</tr>
<tr>
<td>base salaried commission employee: firstName lastName social security number: SSN gross sales: grossSales; commission rate: commissionRate; base salary: baseSalary</td>
</tr>
</tbody>
</table>
```

**Fig. 10.3** | Polymorphic interface for the Employee hierarchy classes.
shows the desired results of each method. We do not list superclass Employee's get and set methods because they're not overridden in any of the subclasses—each of these methods is inherited and used "as is" by each subclass.

Let's consider class Employee's declaration (Fig. 10.4). The class includes a constructor that takes the first name, last name and social security number as arguments (lines 11–16); get methods that return the first name, last name and social security number (lines 25–28, 37–40 and 49–52, respectively); set methods that set the first name, last name and social security number (lines 19–22, 31–34 and 43–46, respectively); method toString (lines 55–60), which returns the String representation of an Employee; and abstract method earnings (line 63), which will be implemented by each of the concrete subclasses. The Employee constructor does not validate its parameters in this example; normally, such validation should be provided.

Why did we decide to declare earnings as an abstract method? It simply does not make sense to provide an implementation of this method in class Employee. We cannot calculate the earnings for a general Employee—we first must know the specific type of Employee to determine the appropriate earnings calculation. By declaring this method abstract, we indicate that each concrete subclass must provide an appropriate earnings implementation and that a program will be able to use superclass Employee variables to invoke method earnings polymorphically for any type of Employee.

```java
// Fig. 10.4: Employee.java
// Employee abstract superclass.

public abstract class Employee {

private String firstName;
private String lastName;
private String socialSecurityNumber;

// three-argument constructor
public Employee( String first, String last, String ssn )
{
    firstName = first;
    lastName = last;
    socialSecurityNumber = ssn;
} // end three-argument Employee constructor

// set first name
public void setFirstName( String first )
{
    firstName = first; // should validate
} // end method setFirstName

// return first name
public String getFirstName()
{
    return firstName;
} // end method getFirstName
```
10.5.2 Concrete Subclass SalariedEmployee

Class SalariedEmployee (Fig. 10.5) extends class Employee (line 4) and overrides abstract method earnings (lines 33–37), which makes SalariedEmployee a concrete class. The class includes a constructor (lines 9–14) that takes a first name, a last name, a social security number and a weekly salary as arguments; a set method to assign a new nonnegative value to instance variable weeklySalary (lines 17–24); a get method to return weeklySalary’s value (lines 27–30); a method earnings (lines 33–37) to calculate a SalariedEmployee’s earnings; and a method toString (lines 40–45), which returns a String including the employee’s type, namely, "salaried employee; " followed by employee-specific information produced by superclass Employee’s toString method and SalariedEmployee’s getWeeklySalary method. Class SalariedEmployee’s constructor passes the first name, last name and social security number to the Employee constructor (line 12) to initialize the private instance variables not inherited from the superclass. Method earn-

```java
30     // set last name
31     public void setLastName( String last )
32     {
33         lastName = last; // should validate
34     } // end method setLastName
35
36     // return last name
37     public String getLastName()
38     {
39         return lastName;
40     } // end method getLastName
41
42     // set social security number
43     public void setSocialSecurityNumber( String ssn )
44     {
45         socialSecurityNumber = ssn; // should validate
46     } // end method setSocialSecurityNumber
47
48     // return social security number
49     public String getSocialSecurityNumber()
50     {
51         return socialSecurityNumber;
52     } // end method getSocialSecurityNumber
53
54     // return String representation of Employee object
55     @Override
56     public String toString()
57     {
58         return String.format("%s %s
social security number: %s",
59             getFirstName(), getLastName(), getSocialSecurityNumber() );
60     } // end method toString
61
62     // abstract method overridden by concrete subclasses
63     public abstract double earnings(); // no implementation here
64 } // end abstract class Employee
```

Fig. 10.4 | Employee abstract superclass. (Part 2 of 2.)
ings overrides Employee’s abstract method earnings to provide a concrete implementation that returns the SalariedEmployee’s weekly salary. If we do not implement earnings, class SalariedEmployee must be declared abstract—otherwise, class SalariedEmployee will not compile. Of course, we want SalariedEmployee to be a concrete class in this example.

```java
public class SalariedEmployee extends Employee {
    private double weeklySalary;

    // four-argument constructor
    public SalariedEmployee(String first, String last, String ssn,
                             double salary) {
        super(first, last, ssn); // pass to Employee constructor
        setWeeklySalary(salary); // validate and store salary
    }

    // set salary
    public void setWeeklySalary(double salary) {
        if (salary >= 0.0)
            baseSalary = salary;
        else
            throw new IllegalArgumentException(
                "Weekly salary must be >= 0.0");
    }

    // return salary
    public double getWeeklySalary() {
        return weeklySalary;
    }

    // calculate earnings; override abstract method earnings in Employee
    @Override
    public double earnings() {
        return getWeeklySalary();
    }

    // return String representation of SalariedEmployee object
    @Override
    public String toString() {
        return String.format("salaried employee: %s\n%s: $%,.2f",
                            super.toString(), "weekly salary", getWeeklySalary());
    }
}
```

Fig. 10.5 | SalariedEmployee concrete class extends abstract class Employee.
Method `toString` (lines 40–45) overrides `Employee` method `toString`. If class `SalariedEmployee` did not override `toString`, `SalariedEmployee` would have inherited the `Employee` version of `toString`. In that case, `SalariedEmployee`'s `toString` method would simply return the employee's full name and social security number, which does not adequately represent a `SalariedEmployee`. To produce a complete String representation of a `SalariedEmployee`, the subclass's `toString` method returns "salaried employee: " followed by the superclass `Employee`-specific information (i.e., first name, last name and social security number) obtained by invoking the superclass's `toString` method (line 44)—this is a nice example of code reuse. The String representation of a `SalariedEmployee` also contains the employee's weekly salary obtained by invoking the class's `getWeeklySalary` method.

### 10.5.3 Concrete Subclass HourlyEmployee

Class `HourlyEmployee` (Fig. 10.6) also extends `Employee` (line 4). The class includes a constructor (lines 10–16) that takes as arguments a first name, a last name, a social security number, an hourly wage and the number of hours worked. Lines 19–26 and 35–42 declare `set` methods that assign new values to instance variables `wage` and `hours`, respectively. Method `setWage` (lines 19–26) ensures that wage is nonnegative, and method `setHours` (lines 35–42) ensures that hours is between 0 and 168 (the total number of hours in a week) inclusive. Class `HourlyEmployee` also includes `get` methods (lines 29–32 and 45–48) to return the values of wage and hours, respectively; a method `earnings` (lines 51–58) to calculate an `HourlyEmployee`'s earnings; and a method `toString` (lines 61–67), which returns a String containing the employee's type ("hourly employee: ") and the employee-specific information. The `HourlyEmployee` constructor, like the `SalariedEmployee` constructor, passes the first name, last name and social security number to the superclass `Employee` constructor (line 13) to initialize the private instance variables. In addition, method `toString` calls superclass method `toString` (line 65) to obtain the `Employee`-specific information (i.e., first name, last name and social security number)—this is another nice example of code reuse.

---

```java
1 // Fig. 10.6: HourlyEmployee.java
2 // HourlyEmployee class extends Employee.
3
4 public class HourlyEmployee extends Employee {
5     private double wage; // wage per hour
6     private double hours; // hours worked for week
7
8     // five-argument constructor
9     public HourlyEmployee( String first, String last, String ssn,
10         double hourlyWage, double hoursWorked )
11     {
12         super( first, last, ssn );
13         setWage( hourlyWage ); // validate hourly wage
14         setHours( hoursWorked ); // validate hours worked
15     } // end five-argument HourlyEmployee constructor
16 }
```

---

**Fig. 10.6** | HourlyEmployee class extends Employee. (Part 1 of 2.)
public void setWage(double hourlyWage) {
    if (hourlyWage >= 0.0)
        wage = hourlyWage;
    else
        throw new IllegalArgumentException("Hourly wage must be >= 0.0");
} // end method setWage

// return wage
public double getWage() {
    return wage;
} // end method getWage

// set hours worked
public void setHours(double hoursWorked) {
    if ((hoursWorked >= 0.0) && (hoursWorked <= 168.0))
        hours = hoursWorked;
    else
        throw new IllegalArgumentException("Hours worked must be >= 0.0 and <= 168.0");
} // end method setHours

// return hours worked
public double getHours() {
    return hours;
} // end method getHours

// calculate earnings; override abstract method earnings in Employee
@override
public double earnings() {
    if (getHours() <= 40) // no overtime
        return getWage() * getHours();
    else
        return 40 * getWage() + (getHours() - 40) * getWage() * 1.5;
} // end method earnings

// return String representation of HourlyEmployee object
@override
public String toString() {
    return String.format("hourly employee: %s
%s: $%,.2f; %s: %,.2f", super.toString(), "hourly wage", getWage(), "hours worked", getHours());
} // end method toString

Fig. 10.6 | HourlyEmployee class extends Employee. (Part 2 of 2.)
10.5.4 Concrete Subclass CommissionEmployee

Class CommissionEmployee (Fig. 10.7) extends class Employee (line 4). The class includes a constructor (lines 10–16) that takes a first name, a last name, a social security number, a sales amount and a commission rate; set methods (lines 19–26 and 35–42) to assign new values to instance variables commissionRate and grossSales, respectively; get methods (lines 29–32 and 45–48) that retrieve the values of these instance variables; method earnings (lines 51–55) to calculate a CommissionEmployee's earnings; and method toString (lines 58–65), which returns the employee's type, namely, "commission employee: " and employee-specific information. The constructor also passes the first name, last name and social security number to Employee's constructor (line 13) to initialize Employee's private instance variables. Method toString calls superclass method toString (line 62) to obtain the Employee-specific information (i.e., first name, last name and social security number).

```java
    // Fig. 10.7: CommissionEmployee.java
    // CommissionEmployee class extends Employee.
    public class CommissionEmployee extends Employee {
        private double grossSales; // gross weekly sales
        private double commissionRate; // commission percentage

        // five-argument constructor
        public CommissionEmployee( String first, String last, String ssn,
                                     double sales, double rate )
        {
            super( first, last, ssn );
            setGrossSales( sales );
            setCommissionRate( rate );
        } // end five-argument CommissionEmployee constructor

        // set commission rate
        public void setCommissionRate( double rate )
        {
            if ( rate > 0.0 && rate < 1.0 )
                commissionRate = rate;
            else
                throw new IllegalArgumentException(                           
                    "Commission rate must be > 0.0 and < 1.0" );
        } // end method setCommissionRate

        // return commission rate
        public double getCommissionRate()
        {
            return commissionRate;
        } // end method getCommissionRate

        // set gross sales amount
        public void setGrossSales( double sales )
        {

    Fig. 10.7  |  CommissionEmployee class extends Employee. (Part 1 of 2.)
```
10.5.5 Indirect Concrete Subclass BasePlusCommissionEmployee

Class BasePlusCommissionEmployee (Fig. 10.8) extends class CommissionEmployee (line 4) and therefore is an indirect subclass of class Employee. Class BasePlusCommissionEmployee has a constructor (lines 9–14) that takes as arguments a first name, a last name, a social security number, a sales amount, a commission rate and a base salary. It then passes all of these except the base salary to the CommissionEmployee constructor (line 12) to initialize the inherited members. BasePlusCommissionEmployee also contains a set method (lines 17–24) to assign a new value to instance variable baseSalary and a get method (lines 27–30) to return baseSalary's value. Method earnings (lines 33–37) calculates a BasePlusCommissionEmployee's earnings. Line 36 in method earnings calls superclass CommissionEmployee's earnings method to calculate the commission-based portion of the employee's earnings—this is another nice example of code reuse. BasePlusCommissionEmployee's toString method (lines 40–46) creates a String representation of a BasePlusCommissionEmployee that contains "base-salaried", followed by the String obtained by invoking superclass CommissionEmployee's toString method (another example of code reuse), then the base salary. The result is a String beginning with "base-salaried commission employee" followed by the rest of the BasePlusCommissionEmployee's information. Recall that CommissionEm-

```java
if (sales >= 0.0)
    grossSales = sales;
else
    throw new IllegalArgumentException("Gross sales must be >= 0.0");
// end method setGrossSales

// return gross sales amount
public double getGrossSales()
{
    return grossSales;
} // end method getGrossSales

// calculate earnings; override abstract method earnings in Employee
@Override
public double earnings()
{
    return getCommissionRate() * getGrossSales();
} // end method earnings

// return String representation of CommissionEmployee object
@Override
public String toString()
{
    return String.format("%s: %s
%s: $%,.2f; %s: %.2f",
    "commission employee", super.toString(),
    "gross sales", getGrossSales(),
    "commission rate", getCommissionRate());
} // end method toString
```
ployee's toString obtains the employee's first name, last name and social security number by invoking the toString method of its superclass (i.e., Employee)—yet another example of code reuse. BasePlusCommissionEmployee's toString initiates a chain of method calls that span all three levels of the Employee hierarchy.

```java
// Fig. 10.8: BasePlusCommissionEmployee.java
// BasePlusCommissionEmployee class extends CommissionEmployee.

public class BasePlusCommissionEmployee extends CommissionEmployee
{
    private double baseSalary; // base salary per week

    // six-argument constructor
    public BasePlusCommissionEmployee( String first, String last,
     String ssn, double sales, double rate, double salary )
    {
        super( first, last, ssn, sales, rate );
        setBaseSalary( salary ); // validate and store base salary
    } // end six-argument BasePlusCommissionEmployee constructor

    // set base salary
    public void setBaseSalary( double salary )
    {
        if ( salary >= 0.0 )
            baseSalary = salary;
        else
            throw new IllegalArgumentException(
                "Base salary must be >= 0.0" );
    } // end method setBaseSalary

    // return base salary
    public double getBaseSalary()
    {
        return baseSalary;
    } // end method getBaseSalary

    // calculate earnings; override method earnings in CommissionEmployee
    @Override
    public double earnings()
    {
        return getBaseSalary() + super.earnings();
    } // end method earnings

    // return String representation of BasePlusCommissionEmployee object
    @Override
    public String toString()
    {
        return String.format("%s %s; %s: $%,.2f",
                           "base-salaried", super.toString(),
                           "base salary", getBaseSalary() );
    } // end method toString
} // end class BasePlusCommissionEmployee
```

Fig. 10.8 | BasePlusCommissionEmployee class extends CommissionEmployee.
10.5.6 Polymorphic Processing, Operator instanceof and Downcasting

To test our Employee hierarchy, the application in Fig. 10.9 creates an object of each of the four concrete classes SalariedEmployee, HourlyEmployee, CommissionEmployee and BasePlusCommissionEmployee. The program manipulates these objects nonpolymorphically, via variables of each object’s own type, then polymorphically, using an array of Employee variables. While processing the objects polymorphically, the program increases the base salary of each BasePlusCommissionEmployee by 10%—this requires determining the object’s type at execution time. Finally, the program polymorphically determines and outputs the type of each object in the Employee array. Lines 9–18 create objects of each of the four concrete Employee subclasses. Lines 22–30 output the String representation and earnings of each of these objects nonpolymorphically. Each object’s toString method is called implicitly by printf when the object is output as a String with the %s format specifier.

```java
public class PayrollSystemTest {
    public static void main( String[] args ) {
        // create subclass objects
        SalariedEmployee salariedEmployee =
            new SalariedEmployee( "John", "Smith", "111-11-1111", 800.00 );
        HourlyEmployee hourlyEmployee =
            new HourlyEmployee( "Karen", "Price", "222-22-2222", 16.75, 40 );
        CommissionEmployee commissionEmployee =
            new CommissionEmployee( "Sue", "Jones", "333-33-3333", 10000, .06 );
        BasePlusCommissionEmployee basePlusCommissionEmployee =
            new BasePlusCommissionEmployee( "Bob", "Lewis", "444-44-4444", 5000, .04, 300 );

        System.out.println( "Employees processed individually:\n" );
        System.out.printf( "%s\n%s: $%,.2f\n\n", salariedEmployee, "earned", salariedEmployee.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n", hourlyEmployee, "earned", hourlyEmployee.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n", commissionEmployee, "earned", commissionEmployee.earnings() );
        System.out.printf( "%s\n%s: $%,.2f\n\n", basePlusCommissionEmployee, "earned", basePlusCommissionEmployee.earnings() );

        // create four-element Employee array
        Employee[] employees = new Employee[ 4 ];

        // initialize array with Employees
        employees[ 0 ] = salariedEmployee;
        employees[ 1 ] = hourlyEmployee;
    }
}
```

Fig. 10.9 | Employee hierarchy test program. (Part 1 of 3.)
38 - employees[2] = commissionEmployee;
40 - System.out.println("Employees processed polymorphically:\n");
41 - // generically process each element in array employees
42 - for (Employee currentEmployee : employees)
43 - {
44 -   System.out.println(currentEmployee); // invokes toString
45 -   // determine whether element is a BasePlusCommissionEmployee
46 -   if (currentEmployee instanceof BasePlusCommissionEmployee)
47 -   {
48 -     // downcast Employee reference to BasePlusCommissionEmployee reference
49 -     BasePlusCommissionEmployee employee =
50 -       (BasePlusCommissionEmployee)currentEmployee;
51 -     employee.setBaseSalary(1.10 * employee.getBaseSalary());
52 -     System.out.printf("new base salary with 10\% increase is: \$%,.2f
"
53 -       , employee.getBaseSalary());
54 -   } // end if
55 -   System.out.printf("earned \$%,.2f\n\n", currentEmployee.earnings());
56 - } // end for
57 - System.out.printf("Employee %d is a %s\n", j,
58 -   employees[j].getClass().getName());
59 - } // end main
60 - } // end class PayrollSystemTest

Employees processed individually:
salaried employee: John Smith
  social security number: 111-11-1111
  weekly salary: $800.00
  earned: $800.00

hourly employee: Karen Price
  social security number: 222-22-2222
  hourly wage: $16.75; hours worked: 40.00
  earned: $670.00

commission employee: Sue Jones
  social security number: 333-33-3333
  gross sales: $10,000.00; commission rate: 0.06
  earned: $600.00

base-salaried commission employee: Bob Lewis
  social security number: 444-44-4444

Fig. 10.9 Employee hierarchy test program. (Part 2 of 3.)
Creating the Array of Employees

Line 33 declares employees and assigns it an array of four Employee variables. Line 36 assigns the reference to a SalariedEmployee object to employees[0]. Line 37 assigns the reference to a HourlyEmployee object to employees[1]. Line 38 assigns the reference to a CommissionEmployee object to employees[2]. Line 39 assigns the reference to a BasePlusCommissionEmployee object to employees[3]. These assignments are allowed, because a SalariedEmployee is an Employee, a HourlyEmployee is an Employee, a CommissionEmployee is an Employee and a BasePlusCommissionEmployee is an Employee. Therefore, we can assign the references of SalariedEmployee, HourlyEmployee, CommissionEmployee and BasePlusCommissionEmployee objects to superclass Employee variables, even though Employee is an abstract class.

Polymorphically Processing Employees

Lines 44–65 iterate through array employees and invoke methods toString and earnings with Employee variable currentEmployee, which is assigned the reference to a different Employee in the array on each iteration. The output illustrates that the appropriate methods for each class are indeed invoked. All calls to method toString and earnings are resolved at execution time, based on the type of the object to which currentEmployee refers. This process is known as dynamic binding or late binding. For example, line 46 implicitly invokes method toString of the object to which currentEmployee refers. As a result of dynamic binding, Java decides which class’s toString method to call at execution time rather than at compile time. Only the methods of class Employee can be called via an Em-

| gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00 earned: $500.00 |
| Employees processed polymorphically: |
| salaried employee: John Smith |
| social security number: 111-11-1111 |
| weekly salary: $800.00 |
| earned $800.00 |
| hourly employee: Karen Price |
| social security number: 222-22-2222 |
| hourly wage: $16.75; hours worked: 40.00 |
| earned $670.00 |
| commission employee: Sue Jones |
| social security number: 333-33-3333 |
| gross sales: $10,000.00; commission rate: 0.06 |
| earned $600.00 |
| base-salaried commission employee: Bob Lewis |
| social security number: 444-44-4444 |
| gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00 |
| new base salary with 10% increase is: $330.00 |
| earned $530.00 |
| Employee 0 is a SalariedEmployee |
| Employee 1 is a HourlyEmployee |
| Employee 2 is a CommissionEmployee |
| Employee 3 is a BasePlusCommissionEmployee |

Fig. 10.9 | Employee hierarchy test program. (Part 3 of 3.)
ployee variable (and Employee, of course, includes the methods of class Object). A super-
class reference can be used to invoke only methods of the superclass—the subclass method
implementations are invoked polymorphically.

Performing Type-Specific Operations on BasePlusCommissionEmployee
We perform special processing on BasePlusCommissionEmployee objects—as we encoun-
ter these objects at execution time, we increase their base salary by 10%. When processing
objects polymorphically, we typically do not need to worry about the “specifics,” but to
adjust the base salary, we do have to determine the specific type of Employee object at ex-
ecution time. Line 49 uses the instanceof operator to determine whether a particular Em-
ployee object’s type is BasePlusCommissionEmployee. The condition in line 49 is true if
the object referenced by currentEmployee is a BasePlusCommissionEmployee. This
would also be true for any object of a BasePlusCommissionEmployee subclass because of
the is-a relationship a subclass has with its superclass. Lines 53–54 downcast currentEm-
ployee from type Employee to type BasePlusCommissionEmployee—this cast is allowed
only if the object has an is-a relationship with BasePlusCommissionEmployee. The condi-
tion at line 49 ensures that this is the case. This cast is required if we’re to invoke subclass
BasePlusCommissionEmployee methods getBaseSalary and setBaseSalary on the cur-
rent Employee object—as you’ll see momentarily, attempting to invoke a subclass-only meth-
od directly on a superclass reference is a compilation error.

Common Programming Error 10.3
Assigning a superclass variable to a subclass variable (without an explicit cast) is a com-
pilation error.

Software Engineering Observation 10.4
If a subclass object’s reference has been assigned to a variable of one of its direct or indirect
superclasses at execution time, it’s acceptable to downcast the reference stored in that
superclass variable back to a subclass-type reference. Before performing such a cast, use the
instanceof operator to ensure that the object is indeed an object of an appropriate subclass.

If the instanceof expression in line 49 is true, lines 53–60 perform the special pro-
cessing required for the BasePlusCommissionEmployee object. Using BasePlusCommis-
sionEmployee variable employee, line 56 invokes subclass-only methods getBaseSalary
and setBaseSalary to retrieve and update the employee's base salary with the 10% raise.

Calling earnings Polymorphically
Lines 63–64 invoke method earnings on currentEmployee, which polymorphically calls
the appropriate subclass object’s earnings method. Obtaining the earnings of the Sala-
riedEmployee, HourlyEmployee and CommissionEmployee polymorphically in lines 63–
64 produces the same results as obtaining these employees’ earnings individually in lines
22–27. The earnings amount obtained for the BasePlusCommissionEmployee in lines 63–
64 is higher than that obtained in lines 28–30, due to the 10% increase in its base salary.
Using Reflection to Get Each Employee’s Class Name

Lines 68–70 display each employee’s type as a String, using basic features of Java’s so-called reflection capabilities. Every object knows its own class and can access this information through the `getClass` method, which all classes inherit from class `Object`. Method `getClass` returns an object of type `Class` (from package `java.lang`), which contains information about the object’s type, including its class name. Line 70 invokes `getClass` on the current object to get its runtime class. The result of the `getClass` call is used to invoke `getName` to get the object’s class name.

Avoiding Compilation Errors with Downcasting

In the previous example, we avoided several compilation errors by downcasting an `Employee` variable to a `BasePlusCommissionEmployee` variable in lines 53–54. If you remove the cast operator `(BasePlusCommissionEmployee)` from line 54 and attempt to assign `Employee` variable `currentEmployee` directly to `BasePlusCommissionEmployee` variable `employee`, you’ll receive an “incompatible types” compilation error. This error indicates that the attempt to assign the reference of superclass object `currentEmployee` to subclass variable `employee` is not allowed. The compiler prevents this assignment because a `CommissionEmployee` is not a `BasePlusCommissionEmployee`—the “is-a” relationship applies only between the subclass and its superclasses, not vice versa.

Similarly, if lines 56 and 60 used superclass variable `currentEmployee` to invoke subclass-only methods `getBaseSalary` and `setBaseSalary`, we’d receive “cannot find symbol” compilation errors at these lines. Attempting to invoke subclass-only methods via a superclass variable is not allowed—even though lines 56 and 60 execute only if `instanceof` in line 49 returns true to indicate that `currentEmployee` holds a reference to a `BasePlusCommissionEmployee` object. Using a superclass `Employee` variable, we can invoke only methods found in class `Employee`—`earnings`, `toString` and `Employee`’s `get` and `set` methods.

Software Engineering Observation 10.5

Although the actual method that’s called depends on the runtime type of the object to which a variable refers, a variable can be used to invoke only those methods that are members of that variable’s type, which the compiler verifies.

10.5.7 Summary of the Allowed Assignments Between Superclass and Subclass Variables

Now that you’ve seen a complete application that processes diverse subclass objects polymorphically, we summarize what you can and cannot do with superclass and subclass objects and variables. Although a subclass object also is a superclass object, the two objects are nevertheless different. As discussed previously, subclass objects can be treated as objects of their superclass. But because the subclass can have additional subclass-only members, assigning a superclass reference to a subclass variable is not allowed without an explicit cast—such an assignment would leave the subclass members undefined for the superclass object.

We’ve discussed four ways to assign superclass and subclass references to variables of superclass and subclass types:

1. Assigning a superclass reference to a superclass variable is straightforward.
2. Assigning a subclass reference to a subclass variable is straightforward.
3. Assigning a subclass reference to a superclass variable is safe, because the subclass object *is an* object of its superclass. However, the superclass variable can be used to refer *only* to superclass members. If this code refers to subclass-only members through the superclass variable, the compiler reports errors.

4. Attempting to assign a superclass reference to a subclass variable is a compilation error. To avoid this error, the superclass reference must be cast to a subclass type explicitly. At execution time, if the object to which the reference refers is *not* a subclass object, an exception will occur. (For more on exception handling, see Chapter 11.) You should use the `instanceof` operator to ensure that such a cast is performed only if the object is a subclass object.

### 10.6 final Methods and Classes

We saw in Sections 6.3 and 6.9 that variables can be declared `final` to indicate that they cannot be modified after they’re initialized—such variables represent constant values. It’s also possible to declare methods, method parameters and classes with the `final` modifier.

**Final Methods Cannot Be Overridden**

A `final` method in a superclass *cannot* be overridden in a subclass—this guarantees that the `final` method implementation will be used by all direct and indirect subclasses in the hierarchy. Methods that are declared `private` are implicitly `final`, because it’s not possible to override them in a subclass. Methods that are declared `static` are also implicitly `final`. A `final` method’s declaration can never change, so all subclasses use the same method implementation, and calls to `final` methods are resolved at compile time—this is known as static binding.

**Final Classes Cannot Be Superclasses**

A `final` class that’s declared `final` cannot be a superclass (i.e., a class cannot extend a `final` class). All methods in a `final` class are implicitly `final`. Class `String` is an example of a `final` class. If you were allowed to create a subclass of `String`, objects of that subclass could be used wherever `Strings` are expected. Since class `String` cannot be extended, programs that use `Strings` can rely on the functionality of `String` objects as specified in the Java API. Making the class `final` also prevents programmers from creating subclasses that might bypass security restrictions. For more insights on the use of keyword `final`, visit

download.oracle.com/javase/tutorial/java/IandI/final.html

and


**Common Programming Error 10.5**

Attempting to declare a subclass of a `final` class is a compilation error.

**Software Engineering Observation 10.6**

*In the Java API, the vast majority of classes are not declared `final`. This enables inheritance and polymorphism. However, in some cases, it’s important to declare classes `final`—typically for security reasons.*
10.7 Case Study: Creating and Using Interfaces

Our next example (Figs. 10.11–10.15) reexamines the payroll system of Section 10.5. Suppose that the company involved wishes to perform several accounting operations in a single accounts payable application—in addition to calculating the earnings that must be paid to each employee, the company must also calculate the payment due on each of several invoices (i.e., bills for goods purchased). Though applied to unrelated things (i.e., employees and invoices), both operations have to do with obtaining some kind of payment amount. For an employee, the payment refers to the employee’s earnings. For an invoice, the payment refers to the total cost of the goods listed on the invoice. Can we calculate such different things as the payments due for employees and invoices in a single application polymorphically? Does Java offer a capability requiring that unrelated classes implement a set of common methods (e.g., a method that calculates a payment amount)? Java interfaces offer exactly this capability.

**Standardizing Interactions**

Interfaces define and standardize the ways in which things such as people and systems can interact with one another. For example, the controls on a radio serve as an interface between radio users and a radio’s internal components. The controls allow users to perform only a limited set of operations (e.g., change the station, adjust the volume, choose between AM and FM), and different radios may implement the controls in different ways (e.g., using push buttons, dials, voice commands). The interface specifies what operations a radio must permit users to perform but does not specify how the operations are performed.

**Software Objects Communicate Via Interfaces**

Software objects also communicate via interfaces. A Java interface describes a set of methods that can be called on an object to tell it, for example, to perform some task or return some piece of information. The next example introduces an interface named Payable to describe the functionality of any object that must be capable of being paid and thus must offer a method to determine the proper payment amount due. An interface declaration begins with the keyword `interface` and contains only constants and abstract methods. Unlike classes, all interface members must be public, and interfaces may not specify any implementation details, such as concrete method declarations and instance variables. All methods declared in an interface are implicitly public abstract methods, and all fields are implicitly public, static and final. [Note: As of Java SE 5, it became a better programming practice to declare sets of constants as enumerations with keyword `enum`. See Section 6.9 for an introduction to `enum` and Section 8.9 for additional `enum` details.]

**Using an Interface**

To use an interface, a concrete class must specify that it implements the interface and must declare each method in the interface with the signature specified in the interface declaration. To specify that a class implements an interface add the `implements` keyword and the
name of the interface to the end of your class declaration’s first line. A class that does not
implement all the methods of the interface is an abstract class and must be declared
abstract. Implementing an interface is like signing a contract with the compiler that
states, “I will declare all the methods specified by the interface or I will declare my class
abstract.”

Relating Disparate Types
An interface is often used when disparate (i.e., unrelated) classes need to share common
methods and constants. This allows objects of unrelated classes to be processed polymor-
phically—objects of classes that implement the same interface can respond to the same
method calls. You can create an interface that describes the desired functionality, then im-
plement this interface in any classes that require that functionality. For example, in the ac-
counts payable application developed in this section, we implement interface Payable in
any class that must be able to calculate a payment amount (e.g., Employee, Invoice).

Interfaces vs. Abstract Classes
An interface is often used in place of an abstract class when there’s no default implementa-
tion to inherit—that is, no fields and no default method implementations. Like public ab-
stract classes, interfaces are typically public types. Like a public class, a public interface
must be declared in a file with the same name as the interface and the .java file-name ex-
tension.

Tagging Interfaces
We’ll see in Chapter 17, Files, Streams and Object Serialization, the notion of “tagging
interfaces”—empty interfaces that have no methods or constant values. They’re used to
add is-a relationships to classes. For example, in Chapter 17 we’ll discuss a mechanism
called object serialization, which can convert objects to byte representations and can con-
vert those byte representations back to objects. To enable this mechanism to work with
your objects, you simply have to mark them as Serializable by adding implements Se-
rializable to the end of your class declaration’s first line. Then, all the objects of your
class have the is-a relationship with Serializable.

10.7.1 Developing a Payable Hierarchy
To build an application that can determine payments for employees and invoices alike, we
first create interface Payable, which contains method getPaymentAmount that returns a
double amount that must be paid for an object of any class that implements the interface.
Method getPaymentAmount is a general-purpose version of method earnings of the
Employee hierarchy—method earnings calculates a payment amount specifically for an
Employee, while getPaymentAmount can be applied to a broad range of unrelated objects.
After declaring interface Payable, we introduce class Invoice, which implements interface
Payable. We then modify class Employee such that it also implements interface Payable.
Finally, we update Employee subclass SalariedEmployee to “fit” into the Payable hierarchy by renaming SalariedEmployee method earnings as getPaymentAmount.

**Good Programming Practice 10.2**

> When declaring a method in an interface, choose a method name that describes the method’s purpose in a general manner, because the method may be implemented by many unrelated classes.

Classes Invoice and Employee both represent things for which the company must be able to calculate a payment amount. Both classes implement the Payable interface, so a program can invoke method getPaymentAmount on Invoice objects and Employee objects alike. As we’ll soon see, this enables the polymorphic processing of Invoices and Employees required for the company’s accounts payable application.

The UML class diagram in Fig. 10.10 shows the hierarchy used in our accounts payable application. The hierarchy begins with interface Payable. The UML distinguishes an interface from other classes by placing the word “interface” in guillemets (« and ») above the interface name. The UML expresses the relationship between a class and an interface through a relationship known as realization. A class is said to “realize,” or implement, the methods of an interface. A class diagram models a realization as a dashed arrow with a hollow arrowhead pointing from the implementing class to the interface. The diagram in Fig. 10.10 indicates that classes Invoice and Employee each realize (i.e., implement) interface Payable. As in the class diagram of Fig. 10.2, class Employee appears in italics, indicating that it’s an abstract class. Concrete class SalariedEmployee extends Employee and inherits its superclass’s realization relationship with interface Payable.

![Fig. 10.10](image)

**Fig. 10.10**  | Payable interface hierarchy UML class diagram.

### 10.7.2 Interface Payable

The declaration of interface Payable begins in Fig. 10.11 at line 4. Interface Payable contains public abstract method getPaymentAmount (line 6). The method is not explicitly declared public or abstract. Interface methods are always public and abstract, so they do not need to be declared as such. Interface Payable has only one method—interfaces can have any number of methods. In addition, method getPaymentAmount has no parameters, but interface methods can have parameters. Interfaces may also contain fields that are implicitly final and static.
10.7.3 Class Invoice

We now create class Invoice (Fig. 10.12) to represent a simple invoice that contains billing information for only one kind of part. The class declares private instance variables partNumber, partDescription, quantity and pricePerItem (in lines 6–9) that indicate the part number, a description of the part, the quantity of the part ordered and the price per item. Class Invoice also contains a constructor (lines 12–19), get and set methods (lines 22–74) that manipulate the class’s instance variables and a toString method (lines 77–83) that returns a String representation of an Invoice object. Methods setQuantity (lines 46–52) and setPricePerItem (lines 61–68) ensure that quantity and pricePerItem obtain only nonnegative values.

Line 4 indicates that class Invoice implements interface Payable. Like all classes, class Invoice also implicitly extends Object. Java does not allow subclasses to inherit from more than one superclass, but it allows a class to inherit from one superclass and implement as many interfaces as it needs. To implement more than one interface, use a comma-separated list of interface names after keyword implements in the class declaration, as in:

```java
public class ClassName extends SuperclassName implements FirstInterface, SecondInterface, ...
```

Software Engineering Observation 10.7

All objects of a class that implement multiple interfaces have the is-a relationship with each implemented interface type.
partDescription = description;
setQuantity( count ); // validate and store quantity
setPricePerItem( price ); // validate and store price per item
} // end four-argument Invoice constructor

// set part number
public void setPartNumber( String part )
{
    partNumber = part; // should validate
} // end method setPartNumber

// get part number
public String getPartNumber()
{
    return partNumber;
} // end method getPartNumber

// set description
public void setPartDescription( String description )
{
    partDescription = description; // should validate
} // end method setPartDescription

// get description
public String getPartDescription()
{
    return partDescription;
} // end method getPartDescription

// set quantity
public void setQuantity( int count )
{
    if ( count >= 0 )
        quantity = count;
    else
        throw new IllegalArgumentException( "Quantity must be >= 0" );
} // end method setQuantity

// get quantity
public int getQuantity()
{
    return quantity;
} // end method getQuantity

// set price per item
public void setPricePerItem( double price )
{
    if ( price >= 0.0 )
        pricePerItem = price;
    else
        throw new IllegalArgumentException( "Price per item must be >= 0" );
} // end method setPricePerItem

Fig. 10.12  |  Invoice class that implements Payable. (Part 2 of 3.)
Class Invoice implements the one method in interface Payable—method getPaymentAmount is declared in lines 86–90. The method calculates the total payment required to pay the invoice. The method multiplies the values of quantity and pricePerItem (obtained through the appropriate get methods) and returns the result (line 89). This method satisfies the implementation requirement for this method in interface Payable—we’ve fulfilled the interface contract with the compiler.

### 10.7.4 Modifying Class Employee to Implement Interface Payable

We now modify class Employee such that it implements interface Payable. Figure 10.13 contains the modified class, which is identical to that of Fig. 10.4 with two exceptions. First, line 4 of Fig. 10.13 indicates that class Employee now implements interface Payable. So we must rename earnings to getPaymentAmount throughout the Employee hierarchy. As with method earnings in the version of class Employee in Fig. 10.4, however, it does not make sense to implement method getPaymentAmount in class Employee because we cannot calculate the earnings payment owed to a general Employee—we must first know the specific type of Employee. In Fig. 10.4, we declared method earnings as abstract for this reason, so class Employee had to be declared abstract. This forced each Employee concrete subclass to override earnings with an implementation.

In Fig. 10.13, we handle this situation differently. Recall that when a class implements an interface, it makes a contract with the compiler stating either that the class will implement each of the methods in the interface or that the class will be declared abstract. If the latter option is chosen, we do not need to declare the interface methods as abstract in the abstract class—they’re already implicitly declared as such in the interface. Any
concrete subclass of the abstract class must implement the interface methods to fulfill the superclass's contract with the compiler. If the subclass does not do so, it too must be declared abstract. As indicated by the comments in lines 62–63, class Employee of Fig. 10.13 does not implement method getPaymentAmount, so the class is declared abstract. Each direct Employee subclass inherits the superclass's contract to implement method getPaymentAmount and thus must implement this method to become a concrete class for which objects can be instantiated. A class that extends one of Employee's concrete subclasses will inherit an implementation of getPaymentAmount and thus will also be a concrete class.

```java
public abstract class Employee implements Payable {
    private String firstName;
    private String lastName;
    private String socialSecurityNumber;

    // three-argument constructor
    public Employee( String first, String last, String ssn )
    {
        firstName = first;
        lastName = last;
        socialSecurityNumber = ssn;
    } // end three-argument Employee constructor

    // set first name
    public void setFirstName( String first )
    {
        firstName = first; // should validate
    } // end method setFirstName

    // return first name
    public String getFirstName()
    {
        return firstName;
    } // end method getFirstName

    // set last name
    public void setLastName( String last )
    {
        lastName = last; // should validate
    } // end method setLastName

    // return last name
    public String getLastName()
    {
        return lastName;
    } // end method getLastName
```

Fig. 10.13 | Employee class that implements Payable. (Part 1 of 2.)
Chapter 10  Object-Oriented Programming: Polymorphism

10.7.5 Modifying Class SalariedEmployee for Use in the Payable Hierarchy

Figure 10.14 contains a modified SalariedEmployee class that extends Employee and fulfills superclass Employee’s contract to implement Payable method getPaymentAmount. This version of SalariedEmployee is identical to that of Fig. 10.5, but it replaces method earnings with method getPaymentAmount (lines 34–38). Recall that the Payable version of the method has a more general name to be applicable to possibly disparate classes. The remaining Employee subclasses (e.g., HourlyEmployee, CommissionEmployee and Base-PlusCommissionEmployee) also must be modified to contain method getPaymentAmount in place of earnings to reflect the fact that Employee now implements Payable. We leave these modifications as an exercise.

```java
// set social security number
public void setSocialSecurityNumber( String ssn )
{
    socialSecurityNumber = ssn; // should validate
} // end method setSocialSecurityNumber

// return social security number
public String getSocialSecurityNumber()
{
    return socialSecurityNumber;
} // end method getSocialSecurityNumber

// return String representation of Employee object
public String toString()
{
    return String.format( "%s %s
social security number: %s",
                        getFirstName(), getLastName(), getSocialSecurityNumber() );
} // end method toString

// Note: We do not implement Payable method getPaymentAmount here so
// this class must be declared abstract to avoid a compilation error.

Fig. 10.14  |  SalariedEmployee class that implements interface Payable. (Part 1 of 2.)
```

```java
1 // Fig. 10.14: SalariedEmployee.java
2 // SalariedEmployee class extends Employee, which implements Payable.
3 public class SalariedEmployee extends Employee
4{
    private double weeklySalary;
}
```

Fig. 10.14  |  SalariedEmployee class that implements interface Payable method getPaymentAmount. (Part 2 of 2.)
When a class implements an interface, the same \textit{is-a} relationship provided by inheritance applies. Class \texttt{Employee} implements \texttt{Payable}, so we can say that an \texttt{Employee} \textit{is a} \texttt{Payable}. In fact, objects of any classes that extend \texttt{Employee} are also \texttt{Payable} objects. \texttt{SalariedEmployee} objects, for instance, are \texttt{Payable} objects. Objects of any subclasses of the class that implements the interface can also be thought of as objects of the interface type. Thus, just as we can assign the reference of a \texttt{SalariedEmployee} object to a superclass \texttt{Employee} variable, we can assign the reference of a \texttt{SalariedEmployee} object to an inter-

\begin{figure}[h]
\begin{verbatim}
// four-argument constructor
public SalariedEmployee( String first, String last, String ssn,
          double salary )
{
    super( first, last, ssn ); // pass to Employee constructor
    setWeeklySalary( salary ); // validate and store salary
} // end four-argument SalariedEmployee constructor

// set salary
public void setWeeklySalary( double salary )
{
    if ( salary >= 0.0 )
        baseSalary = salary;
    else
        throw new IllegalArgumentException(
            "Weekly salary must be >= 0.0" );
} // end method setWeeklySalary

// return salary
public double getWeeklySalary()
{
    return weeklySalary;
} // end method getWeeklySalary

// calculate earnings; implement interface Payable method that was
// abstract in superclass Employee
@Override
public double getPaymentAmount()
{
    return getWeeklySalary();
} // end method getPaymentAmount

// return String representation of SalariedEmployee object
@Override
public String toString()
{
    return String.format( "salaried employee: %s\n%s: $%,.2f",
                 super.toString(), "weekly salary", getWeeklySalary() );
} // end method toString
} // end class SalariedEmployee
\end{verbatim}
\caption{SalariedEmployee class that implements interface Payable method getPaymentAmount. (Part 2 of 2.)}
\end{figure}
face Payable variable. Invoice implements Payable, so an Invoice object also is a Payable object, and we can assign the reference of an Invoice object to a Payable variable.

Software Engineering Observation 10.8
When a method parameter is declared with a superclass or interface type, the method processes the object received as an argument polymorphically.

Software Engineering Observation 10.9
Using a superclass reference, we can polymorphically invoke any method declared in the superclass and its superclasses (e.g., class Object). Using an interface reference, we can polymorphically invoke any method declared in the interface, its superinterfaces (one interface can extend another) and in class Object—a variable of an interface type must refer to an object to call methods, and all objects have the methods of class Object.

10.7.6 Using Interface Payable to Process Invoices and Employees Polymorphically

PayableInterfaceTest (Fig. 10.15) illustrates that interface Payable can be used to process a set of Invoices and Employees polymorphically in a single application. Line 9 declares payableObjects and assigns it an array of four Payable variables. Lines 12–13 assign the references of Invoice objects to the first two elements of payableObjects. Lines 14–17 then assign the references of SalariedEmployee objects to the remaining two elements of payableObjects. These assignments are allowed because an Invoice is a Payable, a SalariedEmployee is an Employee and an Employee is a Payable. Lines 23–29 use the enhanced for statement to polymorphically process each Payable object in payableObjects, printing the object as a String, along with the payment amount due. Line 27 invokes method toString via a Payable interface reference, even though toString is not declared in interface Payable—all references (including those of interface types) refer to objects that extend Object and therefore have a toString method. (Method toString also can be invoked implicitly here.) Line 28 invokes Payable method getPaymentAmount to obtain the payment amount for each object in payableObjects, regardless of the actual type of the object. The output reveals that the method calls in lines 27–28 invoke the appropriate class’s implementation of methods toString and getPaymentAmount. For instance, when currentPayable refers to an Invoice during the first iteration of the for loop, class Invoice’s toString and getPaymentAmount execute.

```
1 // Fig. 10.15: PayableInterfaceTest.java
2 // Tests interface Payable.
3
4 public class PayableInterfaceTest
5 {
6    public static void main( String[] args )
7    {
8        // create four-element Payable array
9        Payable[] payableObjects = new Payable[ 4 ];
```
10.7.7 Common Interfaces of the Java API

In this section, we overview several common interfaces found in the Java API. The power and flexibility of interfaces is used frequently throughout the Java API. These interfaces are implemented and used in the same manner as the interfaces you create (e.g., interface

```java
// populate array with objects that implement Payable
payableObjects[ 0 ] = new Invoice( "01234", "seat", 2, 375.00 );
payableObjects[ 1 ] = new Invoice( "56789", "tire", 4, 79.95 );
payableObjects[ 2 ] =
    new SalariedEmployee( "John", "Smith", "111-11-1111", 800.00 );
payableObjects[ 3 ] =
    new SalariedEmployee( "Lisa", "Barnes", "888-88-8888", 1200.00 );

System.out.println(
   "Invoices and Employees processed polymorphically:\n" );
// generically process each element in array payableObjects
for ( Payable currentPayable : payableObjects )
{
    // output currentPayable and its appropriate payment amount
    System.out.printf( "%s
%s\n
", currentPayable.toString(),
        "payment due", currentPayable.getPaymentAmount() );
} // end for
} // end main
} // end class PayableInterfaceTest
```

![Fig. 10.15](image-url) | Payable interface test program processing Invoices and Employees polymorphically. (Part 2 of 2.)

Invoices and Employees processed polymorphically:

invoice:
part number: 01234 (seat)
quantity: 2
price per item: $375.00
payment due: $750.00

invoice:
part number: 56789 (tire)
quantity: 4
price per item: $79.95
payment due: $319.80

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
payment due: $800.00

salaried employee: Lisa Barnes
social security number: 888-88-8888
weekly salary: $1,200.00
payment due: $1,200.00
Payable in Section 10.7.2). The Java API’s interfaces enable you to use your own classes within the frameworks provided by Java, such as comparing objects of your own types and creating tasks that can execute concurrently with other tasks in the same program. Figure 10.16 overviews a few of the more popular interfaces of the Java API that we use in Java for Programmers, 2/e.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comparable</td>
<td>Java contains several comparison operators (e.g., &lt;, &lt;=, &gt;, &gt;=, ==, !=) that allow you to compare primitive values. However, these operators cannot be used to compare objects. Interface Comparable is used to allow objects of a class that implements the interface to be compared to one another. Interface Comparable is commonly used for ordering objects in a collection such as an array. We use Comparable in Chapter 18, Generic Collections, and Chapter 19, Generic Classes and Methods.</td>
</tr>
<tr>
<td>Serializable</td>
<td>An interface used to identify classes whose objects can be written to (i.e., serialized) or read from (i.e., deserialized) some type of storage (e.g., file on disk, database field) or transmitted across a network. We use Serializable in Chapter 17, Files, Streams and Object Serialization, and Chapter 24, Networking.</td>
</tr>
<tr>
<td>Runnable</td>
<td>Implemented by any class for which objects of that class should be able to execute in parallel using a technique called multithreading (discussed in Chapter 23, Multithreading). The interface contains one method, run, which describes the behavior of an object when executed.</td>
</tr>
<tr>
<td>GUI event-listener interfaces</td>
<td>You work with graphical user interfaces (GUIs) every day. In your web browser, you might type the address of a website to visit, or you might click a button to return to a previous site. The browser responds to your interaction and performs the desired task. Your interaction is known as an event, and the code that the browser uses to respond to an event is known as an event handler. In Chapter 14, GUI Components: Part 1, and Chapter 22, GUI Components: Part 2, you’ll learn how to build GUIs and event handlers that respond to user interactions. Event handlers are declared in classes that implement an appropriate event-listener interface. Each event-listener interface specifies one or more methods that must be implemented to respond to user interactions.</td>
</tr>
<tr>
<td>SwingConstants</td>
<td>Contains a set of constants used in GUI programming to position GUI elements on the screen. We explore GUI programming in Chapters 14 and 22.</td>
</tr>
</tbody>
</table>

**Fig. 10.16** Common interfaces of the Java API.

### 10.8 Wrap-Up

This chapter introduced polymorphism—the ability to process objects that share the same superclass in a class hierarchy as if they’re all objects of the superclass. The chapter discussed how polymorphism makes systems extensible and maintainable, then demonstrated how to use overridden methods to effect polymorphic behavior. We introduced abstract
classes, which allow you to provide an appropriate superclass from which other classes can inherit. You learned that an abstract class can declare abstract methods that each subclass must implement to become a concrete class and that a program can use variables of an abstract class to invoke the subclasses’ implementations of abstract methods polymorphically. You also learned how to determine an object’s type at execution time. We discussed the concepts of final methods and classes. Finally, the chapter discussed declaring and implementing an interface as another way to achieve polymorphic behavior.

You should now be familiar with classes, objects, encapsulation, inheritance, interfaces and polymorphism—the most essential aspects of object-oriented programming.

In the next chapter, you’ll learn about exceptions, useful for handling errors during a program’s execution. Exception handling provides for more robust programs.
## Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>^</code></td>
<td>logical NOT</td>
</tr>
<tr>
<td><code>,</code></td>
<td>comma</td>
</tr>
<tr>
<td><code>(</code></td>
<td>left brace</td>
</tr>
<tr>
<td><code>)</code></td>
<td>right brace</td>
</tr>
<tr>
<td><code>@</code></td>
<td>annotation</td>
</tr>
<tr>
<td><code>:=</code></td>
<td>flag</td>
</tr>
<tr>
<td><code>?</code></td>
<td>question mark</td>
</tr>
<tr>
<td><code>?::</code></td>
<td>ternary operator</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>//</code></td>
<td>end-of-line comment</td>
</tr>
<tr>
<td><code>/*</code></td>
<td>start of comment</td>
</tr>
<tr>
<td><code>*/</code></td>
<td>end of comment</td>
</tr>
<tr>
<td><code>\</code></td>
<td>backslash escape sequence</td>
</tr>
<tr>
<td><code>%</code></td>
<td>percent</td>
</tr>
<tr>
<td><code>&amp;&amp;</code></td>
<td>boolean logical inclusive</td>
</tr>
<tr>
<td>`</td>
<td></td>
</tr>
<tr>
<td><code>:=</code></td>
<td>equals assignment operator</td>
</tr>
<tr>
<td><code>&amp;</code></td>
<td>AND</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>==</code></td>
<td>equal</td>
</tr>
<tr>
<td><code>&lt;&gt;</code></td>
<td>not equal</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>not equal</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
</tr>
<tr>
<td><code>+=</code></td>
<td>addition assignment operator</td>
</tr>
<tr>
<td><code>-=</code></td>
<td>subtraction assignment operator</td>
</tr>
<tr>
<td><code>+=</code></td>
<td>addition assignment operator</td>
</tr>
<tr>
<td><code>-=</code></td>
<td>subtraction assignment operator</td>
</tr>
<tr>
<td><code>%</code></td>
<td>modulo</td>
</tr>
<tr>
<td><code>&amp;</code></td>
<td>bitwise AND</td>
</tr>
<tr>
<td><code>^</code></td>
<td>bitwise XOR</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>&lt;&lt;</code></td>
<td>left shift</td>
</tr>
<tr>
<td><code>&gt;&gt;</code></td>
<td>right shift</td>
</tr>
<tr>
<td><code>&gt;&gt;=</code></td>
<td>right shift assignment operator</td>
</tr>
<tr>
<td><code>&lt;&lt;=</code></td>
<td>left shift assignment operator</td>
</tr>
<tr>
<td><code>++</code></td>
<td>postincrement</td>
</tr>
<tr>
<td><code>--</code></td>
<td>predecrement</td>
</tr>
<tr>
<td><code>+=</code></td>
<td>increment</td>
</tr>
<tr>
<td><code>--</code></td>
<td>decrement</td>
</tr>
<tr>
<td><code>*</code></td>
<td>multiplication</td>
</tr>
<tr>
<td><code>/</code></td>
<td>division</td>
</tr>
<tr>
<td><code>%</code></td>
<td>remainder</td>
</tr>
<tr>
<td><code>#</code></td>
<td>horiziontal tab escape sequence</td>
</tr>
<tr>
<td><code>\</code></td>
<td>backslash-character escape sequence</td>
</tr>
<tr>
<td><code>&quot;&quot;</code></td>
<td>double-quote escape sequence</td>
</tr>
<tr>
<td><code>\</code></td>
<td>escape sequence</td>
</tr>
<tr>
<td><code>\n</code></td>
<td>newline escape sequence</td>
</tr>
<tr>
<td><code>\r</code></td>
<td>carriage-return escape sequence</td>
</tr>
<tr>
<td><code>\t</code></td>
<td>horizontal tab escape sequence</td>
</tr>
<tr>
<td><code>\f</code></td>
<td>form-feed escape sequence</td>
</tr>
<tr>
<td><code>\</code></td>
<td>backslash-character escape sequence</td>
</tr>
</tbody>
</table>

## Numerics

- `0` flag [A-149](A-149)
- `0` format flag [A-190](A-190)
- `0x` (hexadecimal prefix) [A-1065](A-1065)
- `127.0.0.1` (localhost IP address) [A-819](A-819), [A-909](A-909)

## Abbreviating assignment expressions

### A
- `abs` method of `Math` [A-116](A-116)
- `abstract` method of `ResultSet` [A-878](A-878)
- `abstract` method of `Resultset` [A-542](A-542), [A-543](A-543), [A-545](A-545)
- `abstract` method of `Resultset` [A-542](A-542), [A-543](A-543), [A-545](A-545)
- `abstract` path [A-616](A-616), [A-620](A-620), [A-280](A-280)
- `abstract` data type (ADT) [A-188](A-188)
- `abstract implementation` [A-616](A-616)
- `abstract` keyword [A-261](A-261)
- `abstract` superclass [A-260](A-260), [A-370](A-370)
- `Abstract Window Toolkit` [A-404](A-404)
- `Abstract Window Toolkit` [A-124](A-124)
- `Abstract Button class` [A-420](A-420), [A-701](A-701), [A-706](A-706)
- `addActionListener` method [A-423](A-423)
- `addItemListener` method [A-425](A-425)
- `isMouseMotionListener` method [A-708](A-708)
- `setMnemonic` method [A-706](A-706)
- `setRolloverIcon` method [A-422](A-422)
- `setSelected` method [A-707](A-707)
- `Abstract Collection` [A-616](A-616)
- `Abstract List` [A-616](A-616)
- `Abstract Map` [A-616](A-616)
- `Abstract Queue` [A-616](A-616)
- `Abstract Set` [A-616](A-616)
- `Abstract Table Model` [A-872](A-872), [A-878](A-878)
- `fireTableStructureChanged` method [A-878](A-878)
- `accept` method of `class` [A-811](A-811), [A-818](A-818)
Adapter Classes used to implement event handlers 447
add a web service reference to an application in NetBeans 974
add an event handler in NetBeans 1076
add method
ArrayList<T> 185, 806
ButtonType 429
JFrame 408
JMenu 706
JMenuBar 707
addActionListener method
of class AbstractButton 423
of class JTextField 414
addAll method
Collections 590, 600
List 587
addFirst method of LinkedList 590
addGap method of class GroupLayout, Group 1072
addGap method of class GroupLayout,ParallelGroup 1072
addGap method of class GroupLayout, SequentialGroup 1072
add a web service reference to an application 973
addItemListener method of class
AbstractButton 425
addition compound assignment operator, += 81
addKeyListener method of class Component 450
addLast method of LinkedList 589
addListSelectionListener method of class JList 345
addMouseListener method of class Component 442
addMouseMotionListener method of class Component 442
addPoint method of class Polygon 492, 494
addSeparator method of class JMenu 707
addTab method of class JTabbedPane 721
addTableModelListener method of class TableModel 872
addTrayIcon method of class SystemTray 1088
addWindowListener method of class Window 700
advertisement 933
aggregation in the UML 331
Agile Alliance (www.agilealliance.org) 18
Agile Manifesto (www.agilemanifesto.org) 18
general software development xxiii, 18, 18
.aif file extension 685, 688
.aiff file extension 685, 688
Ajax 943
id attributes for elements 961
Ajax (Asynchronous JavaScript and XML) 16, 956, 957
Ajax-enabled web applications xxii
ajax request 961
Ajax web applications 957
algorithm in Java Collections Framework 590
aligning components in GroupLayout 1072
aligning decimal points in output 1053
alpha software 19
alphabetizing 506
Amazon S3 19
analysis stage of the software life cycle 324
anchor (a) element 664
anchor field of class GridBagConstraints 725
AND (in SQL) 863, 864
Android 6
Android Market 6
app 15
Market 6
operating system 2, 6
smartphone 6
Android for Programmers: An App-Driven Approach 6
angle bracket (>) for XML elements 663
angle brackets (< and >) 623
animated shape 652
animating a series of images 675
animation 657, 668, 680
Animator applet 648
annotation @Override 233
Annotations @GET 980
@PathParam 980
@Produces 980
@WebMethod 968
@WebParam 969
@WebService 968
Path 979
annotations @Resource 951
anonymous inner class 414, 432, 448
anti-aliasing 651
Apache Derby xxii
Apache Software Foundation 5
Apache Tomcat 968
API (application programming interface) 31, 115
API documentation (download.oracle.com/javase/6/docs/api/) 123
API links
Deprecated 1029
Help 1029
Index 1029
Tree 1029
append method of class StringBuilder 521
applet 647, 652, 659, 803
draggable 674
applet .class file 655
Applet class
getAppletContext method 807
getAudioClip method 685
codeBase method 685
getParameter method 804
play method 685
showStatus method 682
applet container 647, 656
Applet Package 124
applet parameter 803
Applet that draws a string 653
applet XHTML element 655
AppletContext interface 803
showDocument method 803, 807
applet-desc element of a JNLP document 664
applets directory
JDK sample applets 648
applets in the public domain 803
appletviewer applet container 647, 649
Applet menu 650
Quit menu item 650
Reload menu item 650
application 23, 24, 41
command-line arguments 118
application programming interface (API) 7, 115
application server 908
Application servers
Apache Tomcat 968
GlassFish 968
JBoss Application Server 968
application-desc element of a JNLP document 664
height attribute 664
main-class attribute 664
name attribute 664
width attribute 664
arc 488, 648
arc angle 488
arc width and arc height for rounded rectangles 487
Arc2D class 469
    CHORD constant 498
    OPEN constant 498
    PIE constant 498
Arc2D.Double class 494
archive files 220
ArcTest applet 648
args parameter 178
argument index 1054, 1060, 1068
argument list 1055
argument promotion 122
argument to a method 25, 44
arithmetic compound assignment operators 81
arithmetic operators 34
arithmetic overflow 301
ArithmeticException class 295, 300
array 540, 806
bounds checking 151
ignoring element zero 152
length instance variable 142
pass an array element to a method 159
pass an array to a method 159
array-access expression 142
array-creation expression 143
array initializer 145
for multidimensional array 168
nested 168
array of one-dimensional arrays 168
ArrayBlockingQueue class 759, 760, 770, 784
size method 761
arraycopy method of class System 181, 182
ArrayIndexOutOfBoundsException class 151, 153, 494
ArrayList<T> generic class 183, 582, 598, 640, 806, 991
add method 185, 806
clear method 183, 186
get method 185
indexOf method 183
remove method 184, 185
size method 185
toString method 642
trimToSize method 184
Arrays class 180
asList method 588, 589
binarySearch method 180
equals method 180
find method 180, 794
sort method 180
toString method 536
arrow 63
arrow key 450
arrowhead in a UML sequence diagram 353
artifact in the UML 1089
ascending order 181
ASC in SQL 859, 860
ascent 482
ASCII (American Standard Code for Information Interchange) character set 105
ASCII character set
Appendix 1024
asList method of Arrays 588, 589
assert statement 315, 1025
assertion 315
AssertionError class 315
Assigning superclass and subclass references to superclass and subclass variables 259
assignment operator, = 33, 38
assignment operators 81
associate right to left 78
association (in the UML) 329, 330, 331, 362, 363
name 330
associativity of operators 35, 38, 84
left to right 38
right to left 35
asterisk (*) SQL wildcard character 856
asynchronous call 352
asynchronous event 301
Asynchronous JavaScript and XML (Ajax) 956
asynchronous request 956
ATM (automated teller machine) case study 319, 324
ATM class (ATM case study) 329, 330, 334, 336, 338, 342, 349, 350, 351, 352, 353, 361
ATM system 324, 325, 327, 328, 338, 342, 360
ATMCaseStudy class (ATM case study) 395
atomic operation 750, 905
attribute 360, 362, 363
compartment in a class diagram 336
declaration in the UML 336, 338
in the UML 4, 43, 329, 333, 334, 336, 338, 341, 368, 369
declaration of an object 4
name in the UML 336
of a class 3
ASCII element in the UML 336
type in the UML 336
.au file extension 685, 688
audio clip 685, 687, 692
AudioClip interface 685
10op method 685
play method 685
stop method 685
Australian Botanical Gardens
author ISBN table of books database 852, 853
author's table of books database 852
auto commit state 905
text 626
auto boxing an int 626
auto boxing 528, 581, 626
AutoCloseable interface 317, 872
close method 317
autoincremented 852, 862
automated teller machine (ATM) 319, 324
user interface 320
automatic driver discovery (JDBC 4) xxii, 869
automatic garbage collection 304
automatic scrolling 435
automatic updating 661
average 70, 72
.file extension 688
await method of interface Condition 777, 781
awaitTermination method of interface ExecutorService 748
AWT (Abstract Window Toolkit) 404
components 405
AWTEvent class 416
B
B conversion character 1060
b conversion character 1060
B2B (business-to-business) transactions 964
background color 476, 478
backing array 588
backslash (\) 29, 1068, 1069
BalanceInquiry class
(ATM case study) 329, 331, 334, 335, 336, 339, 342, 350, 351, 352, 353, 361, 365, 366, 367
Index 1093
BankDatabase class (ATM case study) 329, 332, 334, 342, 344, 349, 350, 351, 352, 353, 354, 361, 363
bar chart 148, 149, 648
bar of a number 526
BASELINE alignment constant in GroupLayout 1072
baseline of the font 480
BasePlusCommissionEmployee class extends CommissionEmployee 272
BasicStroke class 469, 497, 498
CAP_ROUND constant 499
JOIN_ROUND constant 499
batch file 551
behavior 342
of a class 3
of a system 338, 339, 341, 351
beta software 20
bidirectional iterator 588
bidirectional navigability in the UML 361
BigDecimal class 56, 96
documentation (download.oracle.com/javase/6/docs/api/java/math/BigDecimal.html) 96
BigInteger class 786
binary file 541
binary operator 110
binary search algorithm 598
binarySearch method of Arrays 180, 182
of Collections 590, 598, 600
BindException class 818
binding the server to the port 811, 826
BindingProvider interface 1001
getRequestContext method 1001
bitwise operators 107
Blackjack 988
blank line 24
blank target frame 807
B1nkle applet 648
block 68, 76, 811, 829
block increment of a 351inder 696
block until connection received 818
blocked state 739, 745
BlockingQueue interface 760
put method 760, 761
take method 760, 761
body
of a class declaration 25
of a loop 69
of a method 25
of an if statement 35
body XHTML element 655
Bohm, C. 62
BOLD constant of class Font 480
book-title capitalization 403, 420
books database 852
table relationships 855
Boolean attribute in the UML 334
class 580
boolean expression 66, 1042
promotions 123
boolean logical AND, & 107, 109
boolean logical exclusive OR, ^ 107, 110
truth table 110
boolean logical inclusive OR, | 109
boolean primitive type 66, 1025, 1026, 1042
border of a JFrame 699
BorderLayout class 441, 452, 454, 457, 466
CENTER constant 441, 457, 460
EAST constant 441, 457
NORTH constant 441, 457
SOUTH constant 441, 457
WEST constant 441, 457
both constant of class GridBagConstraints 726
either constant of class GridBagConstraints 726
bottom tier 913
bounced buffer 770
bounding rectangle 486, 488, 696
bounds checking 151
Box class 466, 722, 723
createGlue method 725
createHorizontalBox method 466, 723
createHorizontalGlue method 725
createHorizontalStrut method 725
createRigidArea method 725
createVerticalBox method 724
createVerticalGlue method 725
createVerticalStrut method 724
X_AXIS constant 725
Y_AXIS constant 725
boxing conversion 581, 626
BoxLayout class 466, 722
BoxLayout layout manager 722
braces (( and }) 68, 76, 90, 145
not required 102
braille screen reader 405
break 1025
break mode 1039
break statement 102, 105
breakpoint 1037
inserting 1039, 1041
listing 1050
removing 1051
bricks-and-mortar store 933
brightness 478
Brin, Sergey 15
brittle software 245
browse method of class Desktop 1085
browsing 803
buffer 572, 753
buffered I/O 572
BufferedImage class 498
createGraphics method 498
TYPE_INT_RGB constant 498
BufferedInputStream class 572
BufferedOutputStream class 572
flush method 572
BufferedReader class 573
BufferedWriter class 573
building-block approach to creating programs 4
bulk operation 581
business logic 913
business publications 20
business rule 913
business-to-business (B2B) transactions 964
button 400, 420
button label 420
ButtonGroup class 426, 701, 708
add method 429
byte-based stream 541
Byte class 580
byte keyword 1026
byte primitive type 98, 1025
promotions 123
ByteArrayInputStream class 573
ByteArrayOutputStream class 573
bytecode 8, 26
bytecode verifier 9
C
c option of the jar command 662
cache 912
CachedRowSet interface 885
close method 887
calculations 38, 62
calendar class 1059
getInstance method 1060
call-by-reference 161
call-by-value 161
call method of interface Callable 799
call interface Callable 799
call method 799
Callable interface Callable 799
callback function 958
calling method (caller) 41, 49
camera 6
cancel method of class SwingWorker 799
canceler of the jar command 662
CANCEL_OPTION constant of JFileChooser 577
CannotRealizePlayerException exception 690
canRead method of File 543
canWrite method of File 543
CAP_ROUND constant of class BasicStroke 499
capacity method of class String 518
capacity of a String 517
card games 153
card shuffling 594
Card Shuffling and Dealing 29
Ended with Collections 329, 330, 331, 334, 335, 342, 354, 379
casino 125, 130
cast 258, 298, 300, 301, 304, 308, 310
catch to 298, 1025
Cast catch block 298, 298, 300, 301, 304, 308, 310 catch-block 125, 130
catch a subclass exception 303 catch an exception 296 catch block 298, 298, 300, 301, 304, 308, 310 catch clause 298, 1025 catch keyword 298 catch keyword cast 298, 298, 300, 301, 304, 308, 310 catch a subclass exception 303 catch an exception 296 catch block 298, 298, 300, 301, 304, 308, 310 catch blocks 298, 1025 catch block 153 catch handler 316 catch-or-declare requirement 302 cd to change directories 26 cell method of Math 116 cell padding attribute of 954 cell spacing attribute of 954 CENTER constant 954 BorderLayout 441, 457, 460 FlowLayout 457 GridBagConstraints 726 GroupLayout 1072 center mouse button click 445 centered 454 certificate authority 660 chained exception 311 change directories 26, 648 ChangeEvent class 699 ChangeListener interface 699 stateChanged method 699 changing look-and-feel of a Swing-based GUI 715 char array 505 keyword 1025, 1026 primitive type 32, 98 promotions 123 character constant 105 literal 503 character-based stream 541 Character class 503, 524, 580 charValue method 528 digit method 526 forDigit method 526 isDefined method 524 isDigit method 524 isJavaIdentifierPart method 525 isJavaIdentifierStart method 524 isLetter method 526 isLetterOrDigit method 526 isLowerCase method 526 isUpperCase method 526 static conversion methods 527 toLowerCase method 526 toUpperCase method 526 CharArrayReader class 573 CharArrayWriter class 573 charAt method of String 505 of stringBuilder 520 CharSequence interface 536 charValue method of class Character 528 checkbox 420, 426 checkbox label 425 checked exception 302 Checking with assert that a value is within range 315 child window 695, 716, 718, 719 CHORD constant of class Arc2D 498 circular buffer 771 class 3, 336, 342, 346, 360 class keyword 41 class (cont.)
constructor 42, 53, 362 data hiding 48 declaration 24, 653 declare a method 40 default constructor 53 field 47 file 26 get method 196 instance variable 4, 47, 117 instantiating an object 40 name 24, 218, 362 set method 196 class-average problem 70, 74 class cannot extend a final class 278 Class class 253, 277, 409, 878 getName method 253, 277 getResource method 409 class diagram for the ATM system model 332, 356 in the UML 325, 329, 331, 335, 342, 360, 363, 367, 368, 369 .class file 8, 27 separate one for every class 194 .class file extension 685 class hierarchy 225, 261 class instance creation expression 42, 54 class keyword 24, 41, 1025 class library 226, 251 class loader 9, 220, 409 class method 116 class name fully qualified 47 class variable 117, 210 classwide information 210 ClassCastException class 628 Classes
AbstractButton 420, 422, 701, 706 AbstractCollection 616 AbstractList 616 AbstractMap 616 AbstractQueue 616 AbstractSequentialList 616 AbstractSet 616 AbstractTableModel 872, 878 Classes (cont.)
Classes (cont.)

ConcurrentHashMap 784
ConcurrentLinkedDeque 784
ConcurrentSkipListMap 784
ConcurrentSkipListSet 784
Container 405, 435, 454, 462
ContainerAdapter 443
CopyOnWriteArrayList 784
CopyOnWriteArraySet 784
DatagramPacket 826, 848
DatagramSocket 826
DataInputStream 572
DataOutputStream 572
Date 1059
DelayQueue 784
Desktop 1085
Dimension 680
Double 580, 641
DriverManager 869
Ellipse2D 494
Ellipse2D.Float 494
EmptyStackException 604
EnumSet 209
EventListenerList 418
Exception 301
ExecutionException 788
Executors 741
File 542
FileInputStream 541
FileOutputStream 541
FileReader 541, 573
FileWriter 541
FilterInputStream 571
FilterOutputStream 571
Float 580
FlowLayout 408, 454
FocusAdapter 443
Font 425, 469, 480
FontMetrics 469, 482
Formatter 542, 1053
Frame 699
GeneralPath 469, 499

Classes (cont.)

GradientPaint 469, 497
Graphics 448, 469, 494, 498
Graphics2D 469, 494, 498
GridBagConstraints 725, 731
GridBagLayout 722, 725, 727, 731
GridLayout 454, 460
GridLayout 454, 1071
GridLayout.Group 1072
GridLayoutParallelGroup 1072
GridLayout.SequentialGroup 1072
Gson 985
HashMap 608, 804
HashSet 605
Hashtable 608
HyperlinkEvent 808, 810
IllegalArgumentException 763, 778
Image 669
ImageIcon 409, 669, 678, 679
IndexOutOfBoundsException 153
InetAddress 819, 825, 829, 830
InputEvent 438, 445, 450
InputMismatchException 295
InputStream 571, 811, 812, 813
InputStreamReader 573
Integer 403, 580, 641
InterruptedException 742
ItemEvent 425, 429
JApplet 653, 700
JAXB 860
JButton 404, 420, 423, 460
JCheckBox 404, 423
JCheckBoxMenuItem 700, 701, 707
JColorChooser 476
 JComboBox 404, 429, 726

Classes (cont.)

JComponent 405, 406, 408, 418, 429, 433, 446, 462, 469, 471, 680
JdbcRowSetImpl 887
JDesktopPane 716
JDialog 707
JEditorPane 808
JFileChooser 574
JFrame 699
JInternalFrame 716, 718
JLabel 404, 406
JList 404, 433
JMenu 700, 707, 718
JMenuBar 700, 707, 718
JMenuItem 701, 718
OptionPane 401
Panel 404, 446, 447, 454, 462, 675, 696
PasswordField 410, 415
PopupMenu 708
Progressbar 795
RadioButton 423, 426, 429
RadioButtonMenuItem 700, 701, 708
ScrollPane 435, 437, 466, 467
JSlider 695, 696, 699, 1072
JTabbedPane 720, 725
JTable 872
JTextArea 452, 464, 466, 727, 730
JTextComponent 410, 413, 464, 466
JTextField 404, 410, 414, 418, 464
JToggleButton 423
JFrameAdapter 443
KeyEvent 419, 450
Line2D 469, 498
Line2D.Double 494
LinearGradientPaint 497
LineNumberReader 573
LinkedBlockingDeque 784
LinkedBlockingQueue 784
LinkedList 582
LinkedTransferQueue 784
ListSelectionEvent 433
ListSelectionModel 435

Classes (cont.)

Long 580
MalformedURLException 807
Manager 688
Matcher 503, 536
Math 116
MouseAdapter 443
MouseEvent 419, 438, 711
MouseMotionAdapter 443, 447
MouseWheelEvent 439
Number 641
Object 209
ObjectInputStream 541, 812, 813, 819
ObjectOutputStream 541
OutputStream 571, 811, 812, 813
OutputStreamWriter 573
Pattern 503, 536
PipedInputStream 571
PipedOutputStream 571
PipedReader 573
PipedWriter 573
Point 448
Polygon 469, 491
PrintStream 571
PrintWriter 573
PriorityBlockingQueue 784
PriorityQueue 604
Properties 612
RadialGradientPaint 497
Random 124, 125
Reader 573
Rectangle2D 469
Rectangle2D.Double 494
ReentrantLock 777, 779
RoundRect 420
RoundRect 420, Double 494
RowFilter 884
RuntimeException 302
Scanner 32, 45
ServerSocket 811, 818, 840
ServiceManager 672
Short 580
Socket 811, 825, 840, 841
SocketException 826
clear method (cont.) of List<T> 588
of PriorityQueue 604
clearRect method of class Graphics 485
click a button 410
click a tab 652
click count 443
click the mouse 423, 649
click the scroll arrows 432
close method of class 342, 351
of an object 51
client code 257
client connection 811
client-server chat 813
client-server relationship 802
client-side artifacts 973
closePath method of interface 871
close a window 406, 410
close a button 410
class derived from java.awt.Toolkit 649
class derived from java.awt.Toolkit 654
classification 15
classification overview xxiv
Calendar 1073
calendar.currentDayOfWeek 1073
calendar.currentMonth 1073
calendar.currentYear 1073
calendar.dayOfWeek 1073
calendar.dayOfWeek 1073
calendar.isLeapYear 1073
calendar.isLeapYear 1073
calendar.month 1073
calendar.month 1073
calendar.nextDay 1073
calendar.nextDay 1073
calendar.nextMonth 1073
calendar.nextMonth 1073
calendar.nextYear 1073
calendar.nextYear 1073
calendar.previousDay 1073
calendar.previousDay 1073
calendar.previousMonth 1073
calendar.previousMonth 1073
calendar.previousYear 1073
calendar.previousYear 1073
calendar.setTime 1073
calendar.setTime 1073
calendar.setYear 1073
calendar.setYear 1073
classifier 635
classified listing 15
className, this 706
classpath command-line argument 548
to java 221
to javac 220
close debugger command 1050
close method of ArrayList<T> 183

coincidence method in the UML 326
collaboration in the UML 348, 349, 350, 352
collection 183, 579
collection hierarchy 581
collection implementation 615
Collection interface 580, 581, 585, 590
contains method 585
iterator method 585
collections
synchronized collection 582
unmodifiable collection 582
Collections class 582, 625
addAll method 590, 600
binarySearch method 590, 598, 600
copy method 590, 597
disjoint method 590, 600
findFirst method 590, 596
frequency method 590, 600
max method 590, 597
min method 590, 597
reverse method 590, 596
reversOrder method 592
shuffle method 590, 594, 596
sort method 591
wrapper methods 582
collections framework 579
Collections methods reverse, find, copy, max and min 597
collision in a hashtable 609
color 469
color chooser dialog 477
Color class 469
getBlue method 473, 475
getColor method 473
getGreen method 473, 475
getRed method 473, 475
setColor method 473
Color constant 472, 475
color manipulation 471
color switches 478
column 167, 851, 852
column number in a results set 857
columnClasses attribute of h: dataTable 954
columns attribute of h: panelGrid 925
columns of a two-dimensional array 167
com.google.gson.Gson package 985
com.sun.rowset package 887
combo box 400, 429
coma (,) 93
comma (,) formatting flag 95
comma-separated list 93
of arguments 29, 32
of parameters 119
command-and-control software system 736
command button 420
command line 25
command-line argument 118, 178
Command Prompt 8, 25
command window 25, 648, 649, 654
code comment
end-of-line (single-line), // 23, 26
Javadoc 24
single line 26
CommissionEmployee class derived from Employee 270
commit a transaction 905
commit method of interface Connection 905
Common Programming Errors overview xxiv
Commonly used JSF components 922
communication diagram in the UML 326, 351, 352
Comparable<T> interface 290, 509, 591, 625
cmpareTo method 591, 625
Comparator interface 591
compare method 593
Comparator object 591, 597, 606, 608
in sort 591
compare method of interface Comparator 593
compareTo method of class String 507, 509
of Comparable 591
compareTo method of Comparable<T> 625
comparing String objects 506
comparison operator 290
compartment in a UML class diagram 43
compile 26
compile a program 8
compile method of class Pattern 537
compile-time type safety 579
compiled applet class 655
compiler options -d 218
compile-time type safety 619
compiling an application with multiple classes 43
complex curve 499
component 2, 124, 437
Component class 405, 438, 471, 472, 478, 673, 680, 700, 731
addKeyListener method 450
addMouseListener method 442
addMouseMotionListener method 442
getHeight method 673
getMaximumSize method 1072
getMinimumSize method 680, 698, 1072
getPreferredSize method 680, 698, 1072
getWidth method 673
repaint method 448
setBackground method 478
setBounds method 453
setFont method 425
setLocation method 453, 700
setSize method 453, 700
setVisible method 460, 700
component diagram in the UML 1089
component in the UML 1089
ComponentAdapter class 443
ComponentListener interface 443, 454
composite structure diagram in the UML 1090
composition 203, 225, 227, 330, 331, 356
in the UML 330
compound assignment operators 81, 84
compound interest 93
computerized scientific notation 1055
concat method of class String 514
concatenate strings 212
concatenation 120
context class 260
correspondent subclass 265
CONCUR_READ_ONLY constant 877
CONCUR_UPDATABLE constant 877
currency 736
Concurrency API 737
concurrent access to a Collection by multiple threads 615
concurrent collections 736
concurrent debugging 737
concurrent threads 759
ConcurrentHashMap class 784
ConcurrentLinkedDeque class 784
ConcurrentSkipListMap class 784
ConcurrentSkipListSet class 784
Condition 35, 97
Condition interface 777, 779
await method 777, 781
signal method 777
signalAll method 777
condition object 777
conditional AND, && 108, 109
truth table 108
conditional expression 66
conditional operator, ?: 66
conditional OR, || 107, 108
truth table 109
confusing the equality operator == with the assignment operator =
connect to a database 867
connect to server 811, 813
connected lines 491
connected RowSet 885
connection 802, 813, 825, 826, 840, 841
connection between client and server terminates 814
connection between Java program and database 869
Connection interface 869, 871, 876, 905
close method 871
commit method 905
createStatement method 870, 876
getAutoCommit method 905
prepareStatement method 895
rollback method 905
setAutoCommit method 905
connection-oriented service 802
connection-oriented, streams-based transmission 825
connection pool 945
collection port 811
connectedness service 802, 826
connectionless transmission 825
constant 215
consistent state 196
constant state 196
in an interface 290
constant integral expression 98, 105
constant variable 105, 146, 215
must be initialized 146
constructor 42, 53, 362
call another constructor of the same class using this 198
multiple parameters 55
no argument 198
overloaded 195
parameter list 54
Constructor Detail section in API 1034
Constructor Summary section in API 1032
constructors cannot specify a return type 55
consume an event 414
consumer 752
consumer electronic device 7
consumer thread 753
consumer to server 811, 813
consumed a web service 965, 966
continue statement 105, 106, 1025
continuous beta 20
control statement 62, 64, 65
nesting 64
stacking 64
control variable 87, 88, 89
controller (in MVC architecture) 922, 922
controller logic 913
controlling expression of a switch 102
controls 399
conversion characters 1054
\% 1061
A 1056
a 1056
B 1060
b 1060, 1061
C 1057
c 1057
d 1054
E 1055, 1056
e 1055, 1056
f 1055, 1056
G 1056
g 1056
h 1060
H 1060
n 1061
o 1054
S 1057
s 1057
T 1058
cont consumer 910
cont consumer 910
cont debugger command 1040
Container class 405, 435, 454, 462
setLayout method 408, 454, 460, 462, 725
validate method 462
container for menus 700
ContainerAdapter class 443
ContainerListener interface 443
contains method of Collection 585
contains method of class ArrayList<T> 183, 186
containsKey method of Map 611
content pane 435, 708
setBackground method 435
context-sensitive popup menu 708
continue statement 105, 106, 1025
continuous beta 20
control statement 62, 64, 65
nesting 64
stacking 64
control variable 87, 88, 89
controller (in MVC architecture) 922, 922
controller logic 913
controlling expression of a switch 102
controls 399
conversion characters 1054
\% 1061
A 1056
a 1056
B 1060
b 1060, 1061
C 1057
c 1057
d 1054
E 1055, 1056
e 1055, 1056
f 1055, 1056
G 1056
g 1056
h 1060
H 1060
n 1061
o 1054
S 1057
s 1057
T 1058
cont consumer 910
cont consumer 910
cont debugger command 1040
Container class 405, 435, 454, 462
setLayout method 408, 454, 460, 462, 725
validate method 462
container for menus 700
ContainerAdapter class 443
ContainerListener interface 443
contains method of Collection 585
contains method of class ArrayList<T> 183, 186
containsKey method of Map 611
content pane 435, 708
setBackground method 435
context-sensitive popup menu 708
continue statement 105, 106, 1025
continuous beta 20
control statement 62, 64, 65
nesting 64
stacking 64
control variable 87, 88, 89
controller (in MVC architecture) 922, 922
controller logic 913
controlling expression of a switch 102
controls 399
conversion characters 1054
\% 1061
A 1056
a 1056
B 1060
b 1060, 1061
C 1057
c 1057
d 1054
E 1055, 1056
e 1055, 1056
f 1055, 1056
G 1056
g 1056
h 1060
H 1060
n 1061
o 1054
S 1057
s 1057
T 1058
conversion characters

cont.)
t 1058
X 1054
x 1054

cosine of a class

counter-controlled

cursor 25, 28
custom drawing area 447
customized subclass of class

cyclic gradient 497

D

d compiler option 218
dangling-else problem 67
dashed lines 494
data hiding 48
data integrity 203
data source name 947
data tier 913
database 850, 855
table 851
database-driven multitier
web address book xxii
database management
system (DBMS) 850
datagram packet 802, 825, 826
datagram socket 802, 826
DatagramPacket class
826, 848
getAddress method 829
getData method 829
getLength method 829
getPort method 829
DatagramSocket class 826
receive method 829
send method 829
Datatable interface 572
DatatableStream class
572
Dataoutput interface 572
writeBoolean method 572
writeByte method 572
writeBytes method 572
writeChar method 572
writeChars method 572
writeDouble method 572
writeDouble method 572
writeFloat method 572
writeInt method 572
writeLong method 572
writeShort method 572
writeUTF method 572
DatatableoutputStream class
572

DataSource interface 951
date 124
date and time compositions 1058

date class 1059
date formatting 1054
DB2 850
dead state 739
deadlock 778, 781
dealing 153
debugger 1037
break mode 1039
breakpoint 1037
clear command 1050
cont command 1040
defined 1037
exit command 1046
g compiler option 1038
inserting breakpoints 1039
javac command 1039
logic error 1037
next command 1045
print command 1041, 1042
run command 1039, 1041
set command 1041, 1042
step command 1043
step up command 1044
stop command 1039, 1041
suspending program
execution 1041
umwatch command
1046, 1048
watch command 1046
decimal integer 1054
decimal integer formatting 33
decision 35, 64
symbol in the UML 64, 341
declarations
class 24
import 31, 33
method 25
declare a method of a class

40
decrement of a control
variable 87
decrement operator, -- 82
dedicated drawing area 446
deep copy 252
default case in a switch
102, 104, 129
default constructor 53,
201, 232

coordinate system

coordinate suffix characters

1058
A 1058
a 1058
B 1058
b 1058
c 1058
D 1058
d 1058
e 1059
F 1058
H 1059
I 1059
j 1059
k 1059
l 1059
M 1059
m 1058
P 1059
p 1059
R 1058
r 1058
S 1059
T 1058
Y 1059
y 1059
Z 1059

convert

an integral value to a
floating-point value 123
between number systems 526
cookie 934, 935
deletion 935
expiration 935
expiration date 935
header 935
coordinate system 469, 471
coordinates 654
dimensions (0, 0) 469
copy method of
Collections 590, 597
copying objects

depth copy 252
shallow copy 252
CopyonWriterList
class 784
CopyonWriteArraySet
class 784
core package 27
Core Tag Library (JEE)
922, 926
cos method of Math 116
cosine 116
default exception handler 310
default initial value 50
default keyword 1025
default layout of the
content pane 466
default package 47, 216
default upper bound
(object) of a type
parameter 631
default value 50, 85
define a custom drawing
area 447
degree 488
deitel Resource Centers 20
deitQueque class 784
delegation event model 417
delete method of class
StringBuiider 523
DELETE SQL statement
856, 864
deleteCharAt method of
class StringBuilder
523
delimiter for tokens 529
delimiter string 530
demo directory 651
dependent condition 109
deploy a web app 921
deploying a web service 970
deployment diagram in the
UML 1089
deposit class (ATM case
study) 329, 331, 334,
342, 350, 351, 358, 361,
365, 366
depositSlot class (ATM
case study) 329, 330,
331, 334, 342, 351, 362
deprecated link in API 1029
derived class 225
descending order 181
descending sort (DESC) 859
descent 482
descriptive words and
phrases 334, 335
deserialized object 562
design pattern 18
design patterns xxiii
design process 5, 319, 325,
343, 348
design specification 325
design view in Netbeans
1073
desktop class 1085
desktop element of a
JNLP document 664
desktop integration 661
destroy method of
Applet 654, 657
development tool 648
dialog 401
dialog box 401, 706
dialog font 480
dialogInput font 480
diamond in the UML 63
digit 32, 527, 530
digit method of class
Character 526
digital certificate 660
dimension class 680
dir command on
Windows 648
direct superclass 225, 226
directives in the XML
content pane 466
directory 542, 543
disk I/O completion 301
dismiss a dialog 402
dispatch
a thread 739
an event 419
display a line of text 25
display area 655
display monitor 469
display output 38
dispose method of class
Window 699
dispose of interface
WindowConstants 699
distance between values
(random numbers) 129
dithering 649
ditherTest applet 649
divide by zero 11, 295
division 34, 35
division compound
assignment operator, /=
82
DNS (domain name
system) server 909
DNS lookup 909
DO NOTHING_ON_CLOSE
count of interface
WindowConstants 699
do...while repetition
statement 64, 96, 97,
1025
document 695, 716
dollar sign ($) 24
domain name system
(DNS) server 909
Dorsey, Jack 17
dot (.) separator 42, 95,
116, 210, 494
dotted line in the UML 63
double class 580, 641
parseDouble method
658
double equals, == 38
double-precision floating-
point number 56
double primitive type 32,
56, 74, 1025, 1026
promotions 123
double quotes, " 25, 29
double-selection statement
64
doubleValue method of
Number 642
downcast 276
downcasting 258
drag the scroll box 432
draggable applet 661, 674
dragging the mouse to
highlight 466
draw arc 648
draw complex curve 649
draw graphics 653
draw lines and points 649
draw method of class
Graphics2D 497
draw rectangle 659
draw shapes 469
draw3DRect method of
class Graphics 485, 488
drawArc method of class
Graphics 488
drawImage method of class
Graphics 673
draw line 473
drawing on the screen 471
drawLine method of class
Graphics 485
drawOval method of class
Graphics 491, 493
drawPoly method of class
Graphics 491, 493
drawPolyline method of
class Graphics 485, 498
drawRoundRect method of
class Graphics 486
drawString method of
class Graphics 475,
654, 659
DrawTest applet 649, 650
driver class 41
driverManager class 869
getConnection method
869
drop-down list 404, 429
dummy value 74
duplicate of datagram 826
dynamic binding 275
dynamic content 7
dynamic resizing 141
dynamically resizable array
806
E
EAST constant
of class BorderLayout
441, 457
of class GridBagCon-
straints 726
eBay 18
echo character of class
PasswordField 411
echoes a packet back to the
client 826
Eclipse
demonstration video
(www.deitel.com/
books/javafp2) 23
Eclipse
(www.eclipse.org) 8
Eclipse Foundation 5
edit a program 8
editor 8
EL expression 920
element (XML) 663
element of chance 125
elided UML diagram 329
elegant for garbage
collection 213
eliminate resource leaks
305
E11ipse2D class 469
E11ipse2D, Double class
494
E11ipse2D, Float class
494
e1ipsis (…) in a method
parameter list 177
e1se keyword 1025
emacs 8
email 811
embedded system 6
Employee abstract
superclass 265
Employee class hierarchy
test program 273
Employee class that implements Payable
empty statement (a semicolon, ;) 38, 68, 98
empty string 415, 505
empty XML element 664, 925
EmptyStackException class 604
encapsulation 4
cap 497
End key 450
“end of data entry” 74/send-of-file (EOF) 813
indicator 101
key combinations 551
marker 540
end-of-line (single-line) comment, // 23, 26
end-of-stream 813
tag 663
endswith method of class String 510
enhanced for statement 157
ensureCapacity method of class StringBuilder 518
Enter (or Return) key 418, 650, 651
ENTERED constant of nested class EventType 810
entity-relationship diagram 854
enum 133
constant 206
constructor 207
declaration 206
EnumSet class 209
keyword 133, 1025
values method 208
enumeration 133
enumeration constant 133
EnumSet class 209
range method 209
environment variable CLASSPATH 27
PATH 26
EOF (end-of-file) 813
EOFException class 570
equal likelihood 127
equality operator == to compare String objects 507
equality operators 35
equals method
of class Arrays 180
of class Object 252
of class String 507, 509
equalsIgnoreCase method of class String 507, 509
erase 624, 627
e-reader 2
e-reader device 6
Error class 301
escape character 29, 863
ecape sequence 29, 32, 546, 1068, 1069
\, backslash 29
\", double-quote 29
\t, horizontal tab 29
newline, \n 29, 32
event 290, 338, 410, 472
event classes 416
event-dispatch thread (EDT) 471, 785, 818
event driven 410
event-driven process 471
event handler 290, 410
event handling 410, 413, 418
event source 415
event ID 419
event listener 290, 416, 443
adapter class 443
interface 413, 414, 417, 419, 438, 443
event object 416
event registration 414
event source 415, 416
EventListenerList class 418
EventObject class
getsource method 415
EventType nested class
ACTIVATED constant 810
ENTERED constant 810
EXITED constant 810
EventType nested class of HyperlinkEvent 810
exception 152, 293
handler 152
handling 151
parameter 153
Exception class 301
exception handler 298
Exception Handling
multi-catch 316
exception handling
try-with-resources statement 316
exception parameter 298
Exceptions 153
IndexOutofRangeException 153
execute 10
execute an applet in a web browser 652, 656
execute attribute of f ajax 961
execute method
of JdbcTemplate 887
execute method of the Executor interface 741, 744
executeQuery method of PreparedStatement 896
of Statement 870
executeUpdate method of interface PreparedStatement 896
executing an application 12
execution-time error 11
ExecutionException class 788
Executor interface 741
execute method 741, 744
Executors class 741
newCachedThreadPool method 742
ExecutorService interface 741, 799
awaitTermination method 748
shutdown method 744
submit method 799
exists method of File 543
exit debugger command 1046
exit method of class System 304, 551
exit point
of a control statement 64
EXITED constant of nested class EventType 810
executing a for statement 106
exp method of Math 116
expanded submenu 706
expiration date of a cookie 935
explicit conversion 77
exponential format 1054
exponential method 116
exponential notation 1055
exponentiation operator 95
expression 33
extend a class 225
extends keyword 229, 240, 1025
extensibility 257
cXtensible HyperText Markup Language (XHTML) 908, 909, 915
extensible language 42
cXtensible Markup Language (XML) 663, 972
extension mechanism
extending Java with additional class libraries 220
external event 437
f
f option of the jar command 662
f:ajax element 961
f:execute element
execute attribute 961
f:facet JSF element 954
f:render element
execute attribute 961
f:selectItem element 926
f:validateBean element 926
f:validateDoubleRange element 926
f:validateLength element 926
f:validateLongRange element 926
f:validateRegExp element 926
Facebook 15
Facebook 5, 17
Facets (JSF) 915
Facets Tag Library (JSF) 941
Faces servlet 914
fairness policy of a lock 777
false keyword 35, 66, 1025
fatal error 68
fatal logic error 68
fatal runtime error 11
fault tolerant 33, 293
fault-tolerant program 152
feature-complete 20
field 47
default initial value 50
Field Detail section in API 1033
field of a class 135
Field Summary section in API 1032
field width 94, 1054, 1062
file 540
File class 542
canRead method 543
canWrite method 543
exists method 543
File methods 543
Index

File class (cont.)
getAbsolutePath method 543
getName method 543
getParent method 543
getAbsolutePath method 543
isAbsolute method 543
isDirectory method 543
lastModified method 543
length method 543
list method 543
toURI method 692
used to obtain file and directory information 543

file extensions
    .aif 685, 688
    .aiff 685, 688
    .au 685, 688
    .avi 688
    .gif 669
    .jpeg 669
    .jpg 669
    .mid 685, 688
    .mov 688
    .mp3 688
    .mpeg 688
    .mpg 688
    .png 669
    .rmi 685, 688
    .spl 688
    .swf 688
    .wav 685

file folder 651
File methods 543
file processing 541
File.pathSeparator 546
FileContents interface 678
getLength method 673

FileInputStream class 541, 573
FileReader class 541, 573
FILES_AND_DIRECTORIES constant of 3FileChooser 577
FILES_ONLY constant of 3FileChooser 577
FileWriter class 541, 573
fill method of class Arrays 180, 182
    of class Collections 590, 596
    of class Graphics2D 497, 498, 501
fill method of class Arrays 794
fill pattern 498
fill texture 498
fill with color 469
fill13DRect method of class Graphics 485, 488
fill1Arc method of class Graphics 488
    filled-in shape 498
    filled rectangle 473
    filled three-dimensional rectangle 485
fill10val method of class Graphics 449, 485, 488
fillPolygon method of class Graphics 491, 494
fillRect method of class Graphics 473, 485, 498
fillRoundRect method of class Graphics 486
filter a stream 571
FilterInputStream class 571
FilterOutputStream class 571
final
    class 278
classes and methods 278
    keyword 105, 117, 146, 215, 278, 752, 1025
    local variable 432
    method 278
    variable 146
    final state in the UML 63, 339
    final value 88
    finalize method 209, 252
finally
    block 298, 304, 781
    clause 304, 1025
    keyword 298
find method of class Matcher 537
fireTableStructureChanged method of 878
AbstractTableModel 608
first method of 608
SortedSet 608
Fisher-Yates shuffling algorithm 156
five-pointed star 499
fixed text 34
    in a format string 30, 1054
flag value 74
flags 1054, 1064
flash drive 540
Flicker 15
float
    literal suffix F 604
    primitive type 32, 56, 1025, 1026
    primitive type promotions 123
Float class 580
floating-point constant 93
floating-point conversion specifiers 1063
floating-point literal 56
double by default 56
floating-point number 56, 73, 74, 76, 604, 658, 1056
division 77
double precision 56
double primitive type 56
float primitive type 56
single precision 56
floor method of Math 117
flow of control 69, 76
flow of control in the if...else statement 65
FlowLayout class 408, 454, 455
CENTER constant 457
LEFT constant 457
RIGHT constant 457
setAlignment method 457
flush method of class BufferedOutputStream 572
of class Formatter 840
of class ObjectOutputStream 819
focus 411
class for a GUI application 696, 712
FocusAdapter class 443
FocusListener interface 443
font
    manipulation 471
    name 480
    size 480
    style 480
Font class 425, 469, 480
    BOLD constant 480
    getFamily method 479, 482
    getName method 479, 480
    getSize method 479, 480
    getStyle method 479, 482
    isBold method 479, 482
    isPlain method 479, 482
    ITALIC constant 480
    PLAIN constant 480
    font information 469
    font manipulation 471
    font metrics 482
    ascent 484
descent 484
hight 484
leading 484
font style 423
FontMetrics class 469, 482
getAscent method 483
getDescent method 483
getFontMetrics method 482
getHeight method 483
getLeading method 483
footerClass attribute of h: dataTable 954
for attribute of h:message 931
for repetition statement 64, 89, 91, 92, 93, 95, 1025
activity diagram 92
enhanced 157
example 91
header 89
nested 149
forDigit method of class Character 526
foreign key 853, 855
fork/join framework 799
form 923
formal type parameter 623
format method of class Formatter 551, 1070
of class String 190, 1070
format specifiers 30, 1054
%f for floating-point numbers with precision 78
%# 1061
%b 1060
%b 1060
%b for boolean values 111
%c 1057
%d 33, 1054, 1055
%f 1056
%f 1056
%f 58, 1056
%g 1056
%g 1056
%h 1061
%h 1060
%h 1060
%h 1061
%h (line separator) 552
%k 1055
%k 1057
%k 30, 1054, 1057
%x 1055
%x 1055
format string 30, 1054, 1063
formatted output 1060
, (comma) formatting flag 95
%F format specifier 58
– (minus sign) formatting flag 94
0 flag 149, 190
aligning decimal points in output 1053
boolean values 111
conversion character 1054
date and time compositions 1058
date and time conversion suffix characters 1058
dates 1054
exponential format 1054
field width 94, 1054
floating-point numbers 58
grouping separator 95
inserting literal characters 1053
ter in hexadecimal format 1054
integers in octal format 1054
formatted output (cont.)
left justification 1053
left justify 94
precision 58, 1054
right justification 94, 1053
rounding 1053
times 1054
Formatter class 542, 548, 1053, 1069
close 552
method documentation
(Java.sun.com/javase/6/docs/api/java/util/Formatter.html) 1058, 1069
flush method 840
Formatter class 551, 1070
toString method 1070
FormatterClosedException class 552
formatting display formatted data 29
Formatting date and time with conversion character t 1059
Formatting output with class Formatter 1069
forward slash character (/) in end tags 663
Foursquare 15, 18
Fractal applet 649
fragile software 245
ForeColor 74, 90
frame (in the UML) 354
frame class 699
free graphics programs (www.freebyte.com/graphicprograms) 692
FreeTTS 693
frequency method of Collections 590, 600
FROM SQL clause 856
Gson 987
fully qualified class name 47, 218
function key 450
Future interface 799
get method 799
FutureSplash (.sp1) files 688
G
-g command line option to javac 1038
G.L.M. 669
game playing 125
gaming console 6
garbage collection 737
garbage collector 209, 301, 304, 685
general class average problem 73
general path 499
generali ties 527
generalization in the UML 365
generalPath class 469, 499
closePath method 501
lineTo method 500
moveTo method 500
generic class 183, 619, 628
generic collections 321
generic interface 625
generic method 619, 622, 628
generics 580, 619
actual type arguments 623
angle brackets (< and >) 623
default upper bound (of an) of a type parameter 631
eraser 624
formal type parameter 623
method 622
parameterized class 629
parameterized type 629
scope of a type parameter 631
type parameter 623
type parameter section 623
type variable 623
upper bound of a type parameter 626, 627
upper bound of a wildcard 642
wildcard type argument (?) 642
wildcard without an upper bound 644
wildcards 640, 642
gesture 6
get a value 51
@GET annotation 980
GET HTTP request 910
get method of class ArrayList<T> 185
of interface Future 799
of interface List<T> 585
of interface Map 611
get method 51, 196, 202
get request 912
generated
java.sun.com/new2java/8
getAbsolutePath method of class File 543
getActionCommand method of class ActionEvent 415, 423
getAddress method of class DatagramPacket 829
getAscent method of class FontMetrics 483
getAudioClip method of class Applet 685
getAutoCommit method of interface Connection 905
getBlue method of class Color 473, 475
getByte method of class InetSocketAddress 825
getChars method of class String 505
detail StringBuilder 520
getClass method of class Object 409
getClass method of class Object 253, 277
getClassName method of class StackTraceElement 311
getClassName method of class UIManager
LookAndFeelInfo 715
getClickCount method of class MouseEvent 446
getCodeBase method of class Applet 685
getColor method of class Color 473
getColor method of class Graphics 473
getColor method of class TableModel 872, 878
g etColumn method of ResultSetMetaData 878
getColumn method of ResultSetMetaData 870, 878
getColumn method of class TableModel 872, 878
getColumn method of class ResultSetMetaData 878
getColumnName method of TableModel 872, 878
getColumnType method of ResultSetMetaData 871
getConnection method of DriverManager 869
getContentPane method of class JFrame 435
getControlPanelComponent method of interface Player 690
data method of class DatagramPacket 829
defaultSystemTray method of class SystemTray 1088
descent method of class FontMetrics 483
desktop method of class Desktop 1085
eventType method of class HyperlinkEvent 810
getFamily method of class Font 479, 482
getFileName method of class StackTraceElement 311
getFont method of class Graphics 480
getFontMetrics method of class FontMetrics 482
getFontMetrics method of class Graphics 483
green method of class Color 473, 475
green method of class Component 673
height method of class Component 673
height method of class FontMetrics 483
hostname method of class InetAddress 819
icon method of class JLabel 409
iconHeight method of class ImageIcon 673
iconWidth method of class ImageIcon 673
image method of class Image 673
inetAddress method of class Socket 818
inputStream method of class Socket 812, 813
isVisible method of class UIManager 715
instance method of class Calendar 1060
getInt method of class ResultSet 871
getKeyChar method of class KeyEvent 453
getCode method of class KeyEvent 452
getModifiersText method of class KeyEvent 453
getText method of class KeyEvent 453
getLeading method of class FontMetrics 483
getLength method of class DatagramPacket 829
getLength method of interface FileContents 673
getLineNumber method of class StackTraceElement 311
getLocalHost method of class InetAddress 825, 830
getMaximumSize method of class Component 1072
getMessage method of class Throwable 310
getName method of class StackTraceElement 311
getMinimumSize method of class Component 680, 698, 1072
getModifiers method of class InputEvent 453
getName method of class Class 253, 277
getName method of class File 543
getName method of class Font 479, 480
getObjects method of interface ResultSet 871, 878
goToStream method of class Socket 812
getParameter method of class Applet 804
getParent method of class JFrame 415
getPassword method of class JPasswordField 415
getPoint method of class MouseEvent 448
getPort method of class DatagramPacket 829
getPreferredSize method of class Component 680, 698, 1072
getProperty method of class Properties 612
getRed method of class Color 473, 475
getStackTrace method of interface BindingProvider 1001
getResource method of class Class 409
getRow method of interface ResultSet 878
ingRowCount method of interface TableModel 872, 878
getSelectedFile method of class JFileChooser 577
getSelectedIndex method of class JComboBox 432
getSelectedIndex method of class JList 435
getSelectedText method of class JTextComponent 466
getSelectedValues method of class JList 438
getSize method of class Font 479, 480
getSource method of class EventObject 415
getStackTrace method of class Throwable 310
getStateChange method of class ItemEvent 433
getStyle method of class Font 479, 482
getText method of class JLabel 409
text method of class JTextField 708
getBounds method of interface TableModel 872, 878
getVisualComponent method of interface Player 690
getWidth method of class Component 673
getX method of class MouseEvent 442
getY method of class MouseEvent 442
getAction() method of class JFileChooser 409, 669
file extension .gif 669
glass pane 435
GlassFish application server 908, 913, 914, 968
tester web page 971
Good Programming Practices overview xxiv
Google 15
Goggles 15
Maps 16
Storage 19
Gosling, James 7
goTo method of class HyperlinkEvent 810
gradient 497
Gradients class 469, 497
graphical user interface (GUI) 124, 290, 399
design tool 453
graphics 446, 648, 649, 651, 668
Graphic class 448, 469, 471, 494, 654, 657, 658, 673
clearRect method 485
draw3DRect method 485, 488
drawArc method 488
drawLine method 485
drawOval method 485, 488
drawPolygon method 491, 493
drawPolyline method 491, 493
drawRect method 485, 498
drawRoundRect method 486
drawString method 475, 654, 659
fill3DRect method 485, 488
fillRect method 488
fillRect method 488
fillRect method 488
fillRect method 488
Expert Topic overview xxv
Index
Graphics class (cont.)
fillRect method 473, 485, 498
fillRoundRect method 486
getCode method 473
getFont method 480, 480
getFontMetrics method 483
setColor method 498
setFont method 480
graphics context 471
graphics demo 652
graphics in a platform-independent manner 471
Graphics Interchange Format (GIF) 490, 669
Graphics2D class 469, 494, 498, 501
draw method 497
t11 method 497, 498, 501
rotate method 501
setPaint method 497
setStroke method 497
translate method 501
GraphicsTest applet 649
GridLayout applet 649
greed quantifier 534
grid 460
grid for GridBagLayout layout manager 725
GridBagConstraints class 725, 731
anchor field 725
BOTH constant 726
CENTER constant 726
EAST constant 726
gridheight field 727
gridwidth field 727
gridx field 726
gridy field 726
HORIZONTAL constant 726
 instance variables 725
NONE constant 726
NORTH constant 726
NORTHEAST constant 726
NORTHWEST constant 726
RELATIVE constant 731
REMAINDER constant 731
SOUTH constant 726
SOUTHEAST constant 726
GridBagConstraints class (cont.)
SOUTHWEST constant 726
VERTICAL constant 726
weightx field 727
weighty field 727
WEST constant 726
GridBagConstraints constants RELATIVE and REMAINDER 731
GridBagLayout class 722, 725, 727, 731
setConstraints method 731
GridBagLayout layout manager 727
gridheight field of class GridBagConstraints 727
GridLayout class 454, 460
GridLayout containing six buttons 461
gridwidth field of class GridBagConstraints 727
gridx field of class GridBagConstraints 726
gridy field of class GridBagConstraints 726
GROUP BY 856
grid method of class Matcher 538
grouping separator (formatted output) 95
GridLayout class 454, 722, 1071
BASELINE alignment constant 1072
CENTER alignment constant 1072
default layout manager in Netbeans 1071
groups 1072
LEADING aligning components 1072
LEADING alignment constant 1072
parallel layout of GUI components 1071
recommended GUI design guidelines 1072
sequential horizontal orientation 1071
sequential layout of GUI components 1071
spacings between components 1072
GroupLayout class (cont.)
TRAILING alignment constant 1072
GridLayout.Group class 1072
addGap method 1072
GridLayoutParallelGroup class 1072
addGap method 1072
SequentialGroup class 1072
addGap method 1072
Groupon 15, 17
groups in GroupLayout 1072
Gson class 985
code.google.com/p/google-gson/ 983
fromJson method 987
toJson method 985
guard condition in the UML 65, 341
guarantee not a lock 745
GUI (Graphical User Interface) 290
component 399, 648
design tool 453
guide lines (Netbeans) 1074, 1075
guilements ("and") in the UML 55
H
h conversion character 1060
h conversion character 1060
h:body JSF element 916
h:column JSF element 954
h:commandButton element 922, 926
h:dataTable element 926
h:graphicImage element 922, 923
h:head JSF element 916
h:inputText element 922, 925, 931
h:message element 931
h:outputLink element 922, 926, 939, 940
h:outputStyleSheet element 931
h:outputText element 931
h:panelGrid element 922, 924
h:selectItem element 922
h:selectOneMenu element 922, 926
h:selectOneRadio element 922, 926, 939
handle an exception 296
handshake point 811, 825
hardcopy printer 11
has-a relationship 203, 225, 331
hash bucket 609
hash table 605, 609
hashCode method of Object 253
hashing 608
HashMap class 608, 804
keySet method 611
HashSet class 605
HashTable class 608, 609
hash-table collisions 609
hasNext method of class Scanner 101, 551
of interface Iterator 585, 588
hasPrevious method of ListIterator 588
headerClass attribute of h:dataTable 954
headSet method of class TreeSet 607
heavyweight components 405
height 482
height attribute (CSS) 925
height attribute of the applet—desc element 655, 664
height of a rectangle in pixels 473
Help link in API 1029
helper method 104
hexadecimal integer 1054
"hidden" fields 135
hide a dialog 402
HTTP status codes
(www.w3.org/Protocols/rfc2616/rfc2616-sect10.html) 911
hue 478
Hughes, Chris 17
hyperlink 808, 810
HyperlinkEvent class 808, 810
EventType nested class 810
generateType method 810
getURL method 810
HyperlinkListener interface 810
hyperlinkUpdate method 810
hyperlinkUpdate method of interface
HyperlinkListener 810
HyperText Transfer Protocol (HTTP) 803,
909, 911, 934
I
I/O performance
enhancement 572
icon 403
Icon interface 409
id attribute of a JSF
element 925
did attributes for elements
in Ajax requests and responses 961
IDE (integrated
development
environment) 8
identifier 24, 32
identity column 890
IDENTITY keyword (SQL) 890
IDES
NetBeans 964
IEEE 754
(grouper.ieee.org/groups/754/) 1026
IEEE 754 floating point
1026
if single-selection
statement 35, 63, 64, 65,
98, 1025
activity diagram 65
if..else double-selection
statement 64, 65, 76, 98
activity diagram 65
ignoring array element zero
152
IllegalArgumentExcept-
ion class 190
IllegalMonitorStateException class 763,
778
IllegalStateException
class 556
image 696, 668, 692
Image class 669
image map 668, 682
ImageIcon class 409, 669,
678, 679
getIconHeight method
673
getIconWidth method
673
getImage method 673
paintIcon method
679
ImageObserver interface
673
immutable 505
immutable object 212
immutable String object
505
implement an interface
256, 279, 287
implementation-dependent
code 192
implementation of a
function 265
implementation phase 370
implementation process
343, 360
implements 1025
implements keyword
279, 282
implements multiple
interfaces 439
implicit conversion 77
import declaration 31, 33,
47, 217, 1025
increment 93
a control variable 88
expression 106
of a control variable 87
of a for statement 91
operator, ++ 82
increment and decrement
operators 82
infinite postponement
740, 781
infinite repetition 74
indentation 67
independent software
vendor (ISV) 251
index 151
index (substring) 142
Index link in API 1029
index of a JComboBox 431
entire method 142
indexOf method of class
ArrayList<T> 183
indexOf method of class
String 511
IndexOutOfBoundsException
Exception class 597
indirect superclass 225, 226
InetAddress class 819,
825, 829, 830
getName method
825
getHost method
819
getLocalHost method
825, 830
infinite loop 77, 91, 829,
833
infinite recursion 250
infinity symbol 855
information element of
a JNLP document 664
information hiding 4, 48
information tier 913
inheritance 4, 225, 365,
368, 369, 370
examples 226
extends keyword 229,
240
hierarchy 226, 262
hierarchy for university
CommunityMembers
227
multiple 225
single 225
initial method of
Applet 654, 656,
657, 659
initialComponents
generated method in
NetBeans 1076
initial state in the UML
63, 338, 339
initial value of an attribute
336
initial value of control
variable 87
initialization at the
beginning of each
repetition 79
initialize an applet
656
initialize an applet's instance
variables 659
initialize block (static)
937
initialize list 145
initializing
two-dimensional arrays in
declarations 169
initiate an action 701
inline method calls 200
inner class 413, 425, 448, 707
anonymous 432
object of 426
relationship between an inner class and its top-level class 426
INNER JOIN SQL clause 856, 861
innermost set of brackets 152
input data from the keyboard 38
input dialog 401
input/output 542
input/output operation 63
input/output package 124
InputEvent class 438, 445, 450
getModifiers method 453
isAltDown method 446, 453
isControlDown method 453
is_MetaDown method 446, 453
isKeyDown method 453
InputMismatchException class 295, 298
InputStream class 563, 571, 614, 811, 812, 813
read method 673
InputStreamReader class 573
insert method of class StringBuilder 523
INSERT SQL statement 856, 862
inserting literal characters in the output 1053
insertion point 183, 599
instance 3
instance (non-static) method 211
instance of a class 47
instance variable 4, 47, 47, 56, 117
instanceOf operator 276, 1025
instantiating an object of a class 40
int primitive type 32, 74, 82, 98, 1025, 1026
promotions 123
integer 30
array 145
division 73
quotient 34
value 32
Integer class 180, 403, 580, 641
parseInt method 180, 403
integer conversion characters 1054
integer division 34
integers
suffix L 603
integral expression 105
integrated development environment (IDE) 8
intelligent consumer electronic device
interaction between a web service client and a web service 973
interaction diagram in the UML 351
interaction overview diagram in the UML 1090
interactions among objects 348, 352
interest rate 93
interface 256, 280, 288, 870
declaration 279
implementing more than one at a time 439
tagging interface 563
interface keyword 279, 1025
Interfaces 279
ActionListener 414, 419
AppletContext 803
AudioClip 685
AutoCloseable 317, 872
BlockingQueue 760
CachedRowSet 885
Callable 799
CallableStatement 904
ChangeListener 699
CharSequence 536
Collection 580, 581, 590
Comparable 290, 509, 591, 625
Comparator 591
ComponentListener 443
Condition 777, 779
Connection 869, 871, 876
ContainerListener 443
DataInputStream 572
DataOutput 572
Interfaces (cont.)
DataSource 951
Executor 741
ExecutorService 741, 799
FileOpenService 669, 672
FocusListener 443
Future 799
HyperlinkListener 810
Icon 409
ImageObserver 673
ItemListener 425, 708
Iterator 582
JdbcRowSet 885
KeyListener 419, 443, 450
LayoutManager 453, 457
List 580, 588
ListIterator 582
ListSelectionListener 435, 807
Lock 776
Map 580, 608
MouseListener 419
MouseMotionListener 419, 438, 443
MouseWheelListener 439
ObjectInput 562
ObjectOutput 562
Player 688
PreparedStatement 904
PropertyChangeListener 798
Queue 580, 581, 604, 760
RequestContext 1001
ResultSet 870
ResultSetMetaData 870
Rowset 885
Runnable 741, 841, 290
Serializable 290, 563
Set 580, 581, 605
SortedMap 608
SortedSet 606
Statement 871
SwingConstants 409, 699, 290
TableModel 872
WindowConstants 699
Interfaces (cont.)
WindowListener 443, 700, 884
internal frame closable 718
maximizable 718
resizable 718
Internet 803
Internet domain name in reverse order 217
Internet telephony 15
interrupt method of class Thread 742
InterruptedException class 742
invoke a method 52
invokeLater method of class SwingUtilities 818
IOException class 568
IP address 829, 909
of the server 825
iPhone 15, 18
is-a relationship 225, 257
isAbstract method of File 543
isActionKey method of class KeyEvent 453
isAltDown method of class InputEvent 446, 453
isBold method of class Font 479, 482
isCancelLED method of class SwingWorker 794
isControlDown method of class InputEvent 453
isDefined method of class Character 524
isDesktopSupported method of class Desktop 1085
isDigit method of class Character 524
isDirectory method of File 543
isEmpty method of Arraylist 203
Map 612
Stack 604
isEItalic method of class Font 479, 482
isJavaIdentifierPart method of class Character 525
isJavaIdentifierStart method of class Character 524
isLetter method of class Character 526
isLetterOrDigit method of class Character 526
isLowerCase method of class Character 526
isMetaDown method of class InputEvent 446, 453
isPlain method of class Font 479, 482
isPopupTrigger method of class MouseEvent 711
isRunning method of class Timer 679
isSelected method AbstractButton 708 JToggleButton 426
isShiftDown method of class InputEvent 453
isUpperCase method of class Character 526
ITALIC constant of class Font 480
ItemEvent class 425, 429
getStateChange method 433
itemLabel attribute of f:selectItem 926
ItemListener interface 425, 708
itemStateChanged method 425, 426, 708
itemValue attribute of f:selectItem 926
iteration 72
of a loop 87, 106
iteration (looping) of a for loop 152
iterative model 323
iterator 579
Iterator interface 582
hasNext method 585
next method 585
remove method 585
iterator method of Collection 585

J
Jacopini, G. 62
JApplet class 653, 654, 656, 700
destroy method 654
init method 654, 659
paint method 654, 659
JApplet class (cont.)
start method 654, 659
stop method 654
jar command 662
c option 662
f option 662
v option 662
jar element of a JNLP document 664
JAR file 674, 681
Java 2D API 469, 494, 651, 669
Java 2D shapes 494
Java 2D Shapes package
124
Java 3D API 668, 669, 693
Java Abstract Window Toolkit (AWT) package
124
Java Abstract Window Toolkit Event package
124
Java Advanced Imaging API 669
Java API 115, 289
overview 123
Java API documentation download 34
Java API Interfaces 289
Java applet 653
Java Applet Package
124
Java Application Programming Interface (Java API) 7, 31, 115, 123
Java Architecture for XML Binding (JAXB) 980
Java archive (JAR) file 661
Java class library 7, 31, 115
java command 9, 12, 23
-splash option 1083
Java compiler 8
Java Concurrency Package
124
Java Database Connectivity (JDBC) 850
Java DB xxii, 850, 887, 943
Java DB Developer’s Guide 890
Java debugger 1037
Java development environment 8, 9, 10, 648
Java Development Kit (JDK) 26
Java EE 6 908
Java EE 6 tutorial 908
java element of a JNLP document 664
Java-enabled web browser 647
Java Enterprise Edition (Java EE) 2, 908
java extension 8
java file name extension 40
Java fonts
Dialog 480
DialogInput 480
Monospaced 480
SansSerif 480
Serif 480
Java HotSpot compiler 10
Java Image I/O API 669
Java Input/Output Package
124
Java interpreter 26
Java Keywords 1025
Java Language Package
124
Java look and feel Graphics Repository 692
Java look-and-feel repository (java.sun.com/developer/techDocs/hl/repository) 692
Java Media Framework (JMF)
API 669, 688
download 688
Java Media Framework package
124
Java Micro Edition (Java ME) 2
Java Naming and Directory Interface (JNDI) 947
Java Network Launch Protocol (JNLP) 647, 660, 661
Java Networking Package
124
Java Plug-In 647
Java programming language 6
Java Resource Centers at www.deitel.com/ResourceCenters.htm 126
Java SE 6
API documentation
(java.sun.com/javase/6/docs/api/) 123
package overview
(java.sun.com/javase/6/docs/api/overview-summary.html) 123
Java SE 7 105
Automatically Closing Connections, Statements and ResultSets 872
Java SE 7 (cont.)
ConcurrentLinkedDeque 784
fork/join framework 799
LinkedTransferQueue 784
multi-catch 316
new concurrent collections 784
Strings in switch statements 105
try-with-resources statement 316
type inference with the <> notation 585
Java SE Development Kit (JDK) 7, 24
Java Sound API 669, 693
Java Speech API 669, 693
Java Speech API (java.sun.com/products/java-media/speech) 693
Java Standard Edition (Java SE) 2
Java Swing Event Package
125
Java Swing GUI Components Package
125
Java Utilities Package 124
Java Virtual Machine (JVM) 7, 8, 23, 25
Java Web Start 647, 660, 661
automatic updating 661
desktop integration 661
javaws command 664
overview 665
Java website (java.sun.com) 123
JAVA_HOME environment variable 888
java.applet package 124
java.awt class 699
java.awt package 124, 404, 471, 472, 491, 494, 653, 669, 680, 711
java.awt.color package 494
java.awt.event package 124, 125, 416, 418, 443, 453
java.awt.font package 494
java.awt.geom package 124, 494
java.awt.image package 494
java.awt.image.renderable package 494
java.awt.print package 494
java.beans package 798
java.com 647
java.io package 124, 541
java.lang package 33, 116, 124, 229, 252, 503, 741
imported in every Java program 33
java.math package 56
java.net package 124, 802
java.sql package 124, 869, 870
java.util.concurrent package 124, 741, 760, 783, 799
java.awt.concurrent.locks package 776, 777
java.util.prefs package 612
java.util.regex package 503
Java™ Language Specification (java.sun.com/docs/books/jls/) 35
Java2D API 494
Java2D applet 651
Java2D directory 651
JavaBean 916
JavaBean property 916
JavaBeans Specification 916
javac compiler 8, 26
jadoc comment 24
jadoc utility program 24
JavaScript 908
JavaScript Object Notation (JSON) 966
JavaServer Faces (JSF) xxii
JavaServer Pages (JSP)
XML declaration 915
xmlns attributes 915
javax.faces.bean package 661, 669, 672
javax.media package 124, 688
javax.sql package 951
javax.sql.rowset package 885
javax.swing package 125, 399, 401, 409, 418, 420, 466, 476, 653, 669, 699, 715, 718
javax.swing.event package 125, 416, 435, 443, 699
javax.swing.table package 872, 884
JAXB (Java Architecture for XML Binding) 980
JAXB class 980
marshal method 980
unmarshal method 983
JAX-WS 963
JAX-WS 963
JAX-WS package 125
JBoss Application Server (www.jboss.com/products/platforms/application) 968
JButton class 404, 420, 423, 460
JCheckBox buttons and item events 424
JCheckBox class 404, 423
isSelected method 426
JCheckBoxMenuItem class 700, 701, 707
JColorChooser class 476, 478
showDialog method 477
JComboBox class 404, 429, 726
getSelectedIndex method 432
setMaximumRowCount method 432
JComboBox that displays a list of image names 430
JComponent class 405, 406, 408, 429, 433, 446, 462, 469, 471, 680
paintComponent method 446, 469, 679, 696, 698
repaint method 472
setForeground method 708
setOpaque method 446, 449
setToolTipText method 408
jdbc command 1039
JDBC API 850, 867, 904
driver 850, 851
JDBC 4 xxii
JDBC documentation 851
JDBC information (www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html) 851
JDBC Package 124
db:mysql://localhost/books 869
JDBCRowSet interface 885
close method 887
execute method 887
setCommand method 887
setPassword method 887
setUrl method 887
setUsername method 887
JDBCRowSetImpl class 887
JDesktopPane class 716
JDesktopPane documentation (download.oracle.com/javase/6/docs/api/javax/swing/JDesktopPane.html) 719
JDialog class 707
JDIC (Java Desktop Integration Components)
addTrayIcon method of class SystemTray 1088
browse method of class Desktop 1085
Desktop class 1085
getDefaultSystemTray method of class SystemTray 1088
getDesktop method of class Desktop 1085
isDesktopSupported method of class Desktop 1085
mail method of class Desktop 1085
removeTrayIcon method of class SystemTray 1088
-splash command-line option to the java command 1083
splash screen 1083
SplashScreen class 1084
JDIC (cont.)
SystemTray class 1087
TrayIcon class 1088
JDK 7, 26
demo directory 648, 651
JEditorPane class 808
setPage method 810
JFileChooser class 574
CANCEL_OPTION constant 577
FILES_AND_DIRECTORIES constant 577
FILES_ONLY constant 577
getSelectedFile method 577
setFileSelectionMode method 577
showOpenDialog method 577
JFileChooser dialog 574
JFrame class 699
add method 408
EXIT_ON_CLOSE 410
getContentPane method 435
setDefaultCloseOperation method 410, 699
setMenuBar method 700, 707
setSize method 410
setVisible method 410
JFrame.EXIT_ON_CLOSE 410
JInternalFrame class 716, 718
documentation 719
JLabel class 404, 406
documentation 406
getIcon method 409
getText method 409
setHorizontalAlignment method 409
setHorizontalAlignment method 409
setIcon method 409
setText method 409
setVerticalAlignment method 409
setVerticalAlignment method 409
isVisible method 409
JList class 404, 433
addListSelectionListener method 435
JList class (cont.)
getSelectedIndex method 435
getSelectedValues method 438
setFixedCellHeight method 437
setFixedCellWidth method 437
setListData method 438
setSelectionMode method 435
setVisibleRowCount method 435
JMenu class 700, 707, 718
add method 706
addSeparator method 707
JMenuBar class 700, 707, 718
add method 707
JMenuItem class 701, 718
JMenus and mnemonics 701
JMF (Java Media Framework) API 669, 685, 688
JNDI (Java Naming and Directory Interface) 947
JNLP 672, 674, 681
FileOpenService 669, 672
main-class 662
ServiceManager class 672
JNLP (Java Network Launch Protocol) 661
JNLP document 662
applet-desc element 664
application-desc element 664
desktop element 664
documentation 673
information element 664
jar element 664
java element 664
jnlp element 663
offline-allowed element 664
resources element 664
shortcut element 664
title element 664
vendor element 664
jnlp element of a JNLP document 663
codebase attribute 663
href attribute 663
jnlp.jar 673, 681
JOIN_ROUND constant of class BasicStroke 499
joining database tables 853, 861
Joint Photographic Experts Group (JPEG) 409, 669
JOptionPane class 401, 402
constants for message dialogs 404
documentation 403
PLAIN_MESSAGE constant 403
showInputDialog method 402
showMessageDialog method 403
JOptionPane for message dialogs 401
JOptionPane.ERROR_MESSAGE 404
JOptionPane.INFORMATION_MESSAGE 404
JOptionPane.PLAIN_MESSAGE 404
JOptionPane.QUESTION_MESSAGE 404
JOptionPane.WARNING_MESSAGE 404
JPanel class 404, 446, 447, 454, 462, 675, 696
PasswordField class 410, 415
getPassword method 415
JPEG (Joint Photographic Experts Group) 409, 669
.jpg file extension 669
.jpg file extension 669
PopupMenu class 708
show method 711
ProgressBar class 795
RadioButton class 423, 426, 429
RadioButtonMenuItem class 700, 701, 708
JScrollPane class 435, 437, 466, 467
HORIZONTAL_SCROLLBAR_ALWAYS constant 467
HORIZONTAL_SCROLLBAR_AS_NEEDED constant 467
HORIZONTAL_SCROLLBAR.Never constant 467
JSplitPane class (cont.)
setHorizontalScrollBarPolicy method 467
setVerticalScrollBarPolicy method 467
VERTICAL_SCROLLBAR_ALWAYS constant 467
VERTICAL_SCROLLBAR_AS_NEEDED constant 467
VERTICAL_SCROLLBAR_NEVER constant 467
JSplitPane scrollbar policies 466
JSF (cont.)
h:selectOneMenu element 922, 926
h:selectOneRadio element 922, 926, 939
HTML Tag Library 915, 922
javax.faces.bean package 917
@ManagedBean annotation 917, 920
@ResourceScoped annotation 920
resource library 925
resources folder 925
resources library 925
session expire 935
ui:repeat element 941
JSF Expression Language 916
JSF Facelets Tag Library 941
JSF web-application framework 913
JSTL (Java Server Technology Library) 695, 696, 699, 1072
block increment 696
documentation 699
getValue method 699
major tick marks 695
minor tick marks 695
setInverted method 696
setMajorTickSpacing method 699
setPaintTicks method 699
snap-to ticks 695
tick marks 695
JSON (JavaScript Object Notation) 966
JSON (www.json.org) 966
JTabbedPane class 720, 725
addTab method 721
SCROLL_TAB_LAYOUT constant 725
TOP constant 725
JTable class 872
RowFilter 884
setRowFilter method 884
setRowSorter method 884
sorting and filtering xxiii
TableRowSorter 884
long promotions 123
look-and-feel 404, 405, 453, 711
Nimbus 400
Look-and-feel of a Swing-based GUI 712
look-and-feel of an application 404
LookAndFeel1 nested class of class UIManager 715
look1ngAt method of class Matcher 537
lookUp method of class ServiceManager 672
loop 69, 72
body 96
continuation condition 64
counter 87
infinite 77
statement 64, 68
loop-continuation condition 87, 88, 89, 91, 96, 97, 106
loop method of interface AudioClip 685
loopback address 819
looping 72
lowercase letter 24
lowered rectangle 488
\$ command on UNIX 648

M
m-by-n array 167
Mac OS X 8, 25, 551
Macintosh 471
Macintosh AIFF file format 685, 688
Macintosh look-and-feel 712
Macromedia Flash movies (\texttt{.swf}) 688
magnetic tape 540
main method of class Desktop 1085
main method 32, 41
main thread 744
main-class attribute of the applet-desc element 664
main-class specified in an JNLP document 662
major tick marks of class JSlider 695
make your point (game of craps) 130
making decisions 38
 MALformedURLException class 807
@ManagedBean annotation (JSF) 917, 920
Manager class \texttt{688}
createRealized-
Player method \texttt{688}
LIGHTWEIGHT_RENDERER constant 689
setHint method 689
many-to-many relationship 855
many-to-one mapping 608
many-to-one relationship in the UML 332
Map interface 580, 608
containsKey method 611
get method 611
isEmpty method 612
put method 611
size method 612
mappings of SQL types to Java types 871
marsh1 method of class JAXB 980
mashups 16
Matcher class 503, 536
find method 537
group method 538
lookingAt method 537
matches method 537
replaceAll method 537
replaceFirst method 537
matcher method of class Pattern 537
matches method of class Matcher 537
matches method of class Pattern 537
matches method of class String \texttt{530}
matching catch block 298
Math class 95, 116
abs method 116
ceil method 116
cos method 116
E constant 117
exp method 116
floor method 117
log method 117
max method 117
min method 117
PI constant 117
pow method 95, 116, 117
random method 125
sqrt method 116, 117, 122
tan method 117
Matisse GUI designer (Netbeans) 1071
max method of Collections 590, 597
max method of Math 117
maximize a window 406, 719
maximized internal frame 719
maximum attribute of an h:inputText element 932
maxLength attribute of an h:inputText element 932
MDI (Multiple Document Interface) 695, 716
memory buffer 572
memory leak 209, 304
memory-space/execution-time trade-off 609
memory utilization 609
menu 400, 464, 700, 701
menu bar 400, 700, 707
menu item 701, 706
merge in the UML 341
merge records from tables 860
merge symbol in the UML 69
message 52, 654
message dialog 401, 403
types 403
message in the UML 349, 351, 352, 353
message passing in the UML 353
Meta key 445, 446
meta XHTML element 916
metadata 870
metal look-and-feel 695, 712
method 3, 25, 360
local variable 47
parameter 44, 46
parameter list 44
return type 49
signature 138
static 95
method call 4, 119
method declaration 119
Method Detail section in API 1034
method header 41
method overloading 137
method parameter list 177
Method Summary section in API 1033
methods called automatically during applet's execution 656
nested class 413, 715
relationship between an inner class and its top-level class 426
Nested Class Summary section in API 1032
nested control statements 79, 129
Examination-results problem 79
nested for statement 149, 169, 170, 171, 174
nested if...else selection statement 66, 67
nested message in the UML 353
NetBeans 908
add a web service reference to an application 974
code-completion window 919
creating a Java DB database 952
Show Line Numbers 919
Netbeans demonstration video (www.deitel.com/books/javafp2) 23
NetBeans (www.netbeans.org) 8
NetBeans IDE 908, 964, 1073
add an event handler 1076
create a desktop application 973
create a new project 1073
create a web application 967
Design view 1073
GroupLayout 1071
Guide lines 1074, 1075
New JFrame Form dialog 975
New Web Service Client dialog 974
New Web Service dialog 967
Palette window 1073, 1074
Properties window 1073
snap-to alignment grid 1076
Web Application project 966
NetBeans IDE 1071, 1073
**Design** view 1073
*initComponents* auto-generated method 1076
**Source** view 1073
NetBeans Matisse GUI design tool xxiii
NetBeans Matisse GUI designer 1071
network message arrival 301
networking package 124
New JFrame Form dialog 975
new keyword 32, 42, 143, 144, 1025
newScanner(System.in) expression 32
new state 738
new to Java (www.oracle.com/technetwork/topics/newtojava/overview/index.html) 8
New Web Service Client dialog 974
New Web Service dialog 967
newCachedThreadPool method of class Executors 742
newCondition method of interface Lock 777, 779
newline character 28
newline escape sequence, \n 29, 32, 503
next method of Iterator 585
of ResultSet 871
of Scanner 45
nextDouble method of class Scanner 59
nextInt method of class Random 125, 129
nextLine method of class Scanner 44
Nimbus look and feel 400, 712
swing.properties xxi, 401
Nirvanix 19
no-argument constructor 198, 200
non-static class member 210
NONE constant of class GridBagConstraints 726
nonfatal logic error 68
nonfatal runtime error 11
NoPlayerException exception 690
NORTH constant of class BorderLayout 441, 457
NORTH constant of class GridBagConstraints 726
NORTHEAST constant of class GridBagConstraints 726
NORTHWEST constant of class GridBagConstraints 726
NoSuchElementException class 551, 566
note in the UML 63
Notepad 8
notify method of class Object 763
notify method of Object 253
notifyAll method of class Object 763, 766, 767
notifyAll method of Object 253
noun phrase in requirements document 327
null 1025
null keyword 50, 52, 143, 403
null reserved word 85
Number class 641
doubleValue method 642
number systems 526
numeric Classes 580
0
object 2
object (or instance) 4, 351, 654
Object class 209, 225, 229, 570
clone method 252
equals method 252
finalize method 252
getClass method 253, 277, 409
hashCode method 253
notify method 253, 763
notifyAll method 253, 763, 766, 767
toString method 232, 253
wait method 253, 763
object diagram in the UML 1089
object of a derived class 258
object of a derived class is instantiated 250
object-oriented analysis and design (OOAD) 5
object-oriented design (OOD) 319, 325, 327, 337, 360
object-oriented language 5
object-oriented programming (OOP) 2, 5, 225
object serialization 562, 819
ObjectInput interface 562
readObject method 563
ObjectInputStream class 541, 562, 563, 569, 812, 813, 819
ObjectOutput interface 562
writeObject method 562
ObjectOutputStream class 541, 562, 563, 614
close method 568
flush method 819
octal integer 1054
off-by-one error 89
offer method of PriorityQueue 604
offline-allowed element of a JNLP document 664
ON clause 861
one-to-many relationship 855
one-to-one mapping 608
one-*, two- or three-button mouse 445
one-to-many relationship in the UML 332
one-to-one relationship in the UML 332
OOAD (object-oriented analysis and design) 5
OOD (object-oriented design) 319, 325, 327, 337
OOP (object-oriented programming) 5, 225
opaque Swing GUI components 446
open a file 541
OPEN constant of class Arc2D 498
Open Handset Alliance 6
open method of class Desktop 1085
open source 5, 6
open source software xxiii
openFileDialog method of class
File 678
openMultiFileDialog
method of interface
FileChooser 678
openStream method of
File 987
operand 77
operating system 6
operation compartment in
class diagram 342
operation in the UML 43,
329, 342, 346, 362, 363,
368, 369
operation parameter in the
UML 46, 343, 346, 347,
348
operationName attribute of the @WebMethod
annotation 968
operator 33
operator precedence 34
operator precedence chart 78
Operator Precedence
Chart Appendix 1022
operators 34
\&, boolean logical AND 107,
110
\&\&, conditional AND 107,
109
\%\%, remainder assignment
operator 82
\%, prefix increment/ postincrement 82, 83
\%\%, prefix decrement/ postdecrement
operator 82
\%\%, division assignment
operator 82
\&, boolean logical AND 107,
109
\&\&, conditional AND 107,
109
\%\%, remainder assignment
operator 82
\%, prefix increment/ postincrement 82, 83
\%\%, prefix decrement/ postdecrement
operator 82
\%\%, addition assignment
operator 81
\%, assignment 33, 38
\%, subtraction assignment
operator 82
Operators (cont.)
|, boolean logical inclusive OR 107, 109
||, conditional OR 107, 108
|&|, boolean logical AND, &
107, 109
boolean logical exclusive
OR, \^{} 107, 109
boolean logical inclusive
OR, \^{} 107, 110
cast 77
compound assignment
81, 84
conditional AND, \&\& 108, 109
conditional operator, ?: 66
decrement operator, --
82, 83
increment and decre-
ment 82
increment, ++ 82
logical operators 107,
110, 111
multiplication, * 34
multiplicative: *, / and
\%
postfix decrement 82
postfix increment 82
prefix decrement 82
prefix increment 82
remainder, \% 34, 35
subtraction, - 35

P
PaaS (Platform as a Service) 19
pack method of class
Window 719
package 31, 115, 123, 215,
1089
package access 221
package-access members of
a class 222
package-access methods
221
package declaration 216
package diagram in the
UML 1089
package directory names
218
package directory structure
216
package keyword 1025
package name 47
package overview 123
Packages
com.google.gson.Gson
on 985
default package 47
java.applet 124
java.awt 124, 404,
472, 494, 653, 669,
680, 699, 711
java.awt.color 494
java.awt.event 124,
125, 416, 418, 443,
453
java.awt.font 494
java.awt.geom 124,
494
java.awt.image.
renderable 494
java.awt.print 494
java.beans 798
java.io 124, 541
java.lang 33, 116,
124, 229, 252, 503,
741
java.math 56
java.net 124, 802
packages (cont.)
java.sql 124, 869,
870
java.util 31, 124,
125, 183, 640
java.util.concurrent 124,
741, 760, 783, 799
java.util.concurrent.locks 776,
777
java.util.prefs 612
java.util.regex 503
javax.jnlp 661, 669,
672
javax.swing 124, 688
javax.sql 951
javax.sql.rowset 885
javax.swing.event 125,
416, 418, 435,
443, 699
javax.swing.table
872, 884
packet 802, 826
packet-based
communications 802
Page Down key 450
page layout software 503
Page Up key 450
Page, Larry 15
paint method of JApplet
654, 657, 659
Paint object 497
paintComponent method
of class Component
446, 469, 679, 696, 698
painted method of class
ImageIcon 679
panel 462
parallel layout of GUI
components 1071
parallel operations 736
param element 804
parameter 44, 46
parameter in the UML 46,
343, 346, 347, 348
parameter list 44, 54
parameterized class 629
parameterized type 629
parent directory 543
parent window 403, 695,
716
parent window for a dialog
box 706
parent window specified as
null 706
parentheses 25
parseDouble method of Double 658
parseInt method of class Integer 403
parseInt method of Integer 180
partial page update 958
pass an array element to a method 159
pass an array to a method 159
pass-by-reference 161
pass-by-value 159, 161
passing options to a program 179
password 411
@Path annotation 979
PATH environment variable xxx, xxxi, 26
path information 542
path to a resource 910
@PathParam annotation 980
pathSeparator static field of File 546
pattern 494
Pattern class 503, 536
compile method 537
matcher method 537
matches method 537
pattern matching 857
Payable interface declaration 282
Payable interface hierarchy UML class diagram 281
Payable interface test program processing Invioes and Employees polymorphically 288
peek method of class PriorityQueue 604
peek method of class Stack 604
percent (%) SQL wildcard character 857
perform a calculation 38
perform a task 41
perform an action 25
performing operations concurrently 736
persistent data 540
persistent Hashtable 612
personalization 933
photo sharing 15
PHP 19
physical input operation 572
physical output operation 572
PIE constant of class Arc2D 498
pie-shaped arc 498
pipe 571
PipedInputStream class 571
PipedOutputStream class 571
PipedReader class 573
PipedWriter class 573
pixel ("picture element") 469
pixel coordinates 653
PLAF (pluggable look-and-feel) 695
PLAIN constant of class Font 480, 480
PLAIN_MESSAGE 403
Platform as a Service (PaaS) 19, 19
platform dependency 740
play method of class Applet 685
play method of interface AudioClip 685
Player interface 688
getControlPanel-Component method 690
getVisualComponent method 690
start method 690
playing an AudioClip 685
playing audio 685
pluggable look-and-feel (PLAF) 695
pluggable look-and-feel package 405
plus sign (+) indicating public visibility in the UML 360
PNG (Portable Network Graphics) 409, 669
.png file extension 669
point 480, 649
Point class 448
POJO (Plain Old Java Object) 916, 968
poll method of class PriorityQueue 604
polygon 491, 493
Polygon class 469, 491
addPoint method 492, 494
polyline 491
polylines 491
polymorphic processing of collections 582
polymorphic processing of related exceptions 303
polymorphically process Invoices and Employees 288
polymorphism 105, 253, 255, 365, 366, 376
pool of threads 812
pop method of Stack 604
popup trigger event 708, 711
port 811
port number 811, 813, 825, 826, 830
portability 471
Portability Tips overview xxiv
portable 9
portable GUI 125
Portable Network Graphics (PNG) 409, 669
position number 142
positive and negative arc angles 489
positive degrees 488
POST request 911
postback 933
postcondition 314
postdecrement 83
postfix decrement operator 82
postfix increment operator 82, 91
postgresql 850
postincrement 83, 84
pow method of class Math 95, 116, 117
power (exponent) 117
power of 2 larger than 100 68
prebuilt data structures 579
precedence 34, 38, 84
arithmetic operators 35
chart 35, 78
Precedence Chart Appendix 1022
precision 1054, 1055
format of a floating-point number 58
precision of a formatted floating-point number 58
precondition 314
predecrement 82
predefined character class 530
predicate 857
preemptive scheduling 740
Preferences API 612
prefix decrement operator 82
prefix increment operator 82, 84
preincrementing and postincrementing 83
PreparedStatement interface 889, 890, 892, 895, 904
executeQuery method 896
executeUpdate method 896
setString method 889, 896
prepareStatement method of interface Connection 895
presentation logic 913
presentation of a document 915
previous method of ListIterator 588
primary key 851, 855
primitive type 32, 52, 85, 122
boolean 1042
byte 98
char 32, 98
double 32, 56, 74
float 32, 56
int 32, 33, 74, 82, 98
names are keywords 32
passed by value 161
promotions 123
short 98
principal in an interest calculation 93
principle of least privilege 214
print debugger command 1041
print method of System.out 27, 28
print on multiple lines 28
print spooling 753
printArray generic method 623
printf method of System.out 29, 1053
println method of System.out 28
printStackTrace method of class Throwable 310
PrintStream class 571, 614
PrintWriter class 552, 573
priority of a thread 740
PriorityBlockingQueue class 784
remainder 34
remainder compound assignment operator, %= 82
REMAINDER constant of class GridBagConstraints 731
remainer operator, % 34, 35
remove duplicate String 605
remove method of class ArrayList<T> 184, 185
remove method of interface Iterator 585
removeTableModelListener method of interface TableModel 872
removeTrayIcon method of class SystemTray 1088
render attribute of f ajax 961
rendering XHTML in a web browser 911
Reordering output with argument index 1068
repaint method of class Component 448
repaint method of class JComponent 472
repainted 657
repetition 64
counter controlled 76, 79
sentinel controlled 73, 74, 76
repetition statement 62, 64, 68
do...while 64, 96, 97, 97
for 64, 92
while 64, 68, 69, 72, 76, 87
repetition terminates 69
replaceAll method of class Matcher 537
of class String 535
replaceFirst method of class Matcher 537
of class String 535
Representational State Transfer (REST) 963, 965
representing integers in hexadecimal format 1054
representing integers in octal format 1054
request method 911
RequestContext interface 1001
put method 1001
@RequestScoped annotation (JSF) 920
default 920
required attribute of a JSF element 931
requirements 5, 323
requirements document 319, 323, 325
requirements gathering 323
reserved word 64, 1025
false 65
null 50, 52, 85
true 65
resizable array 806
implementation of a List 582
resolution 469
resource leak 209, 304
resource library (JSF) 925
resource-release code 304
resources element of a JNLP document 664
resources folder of a JSF app 925
responses to a survey 150, 152
REST (Representational State Transfer) 963
restart method of class Timer 679
RESTful web services 965
result 857
result set concurrency 877
result set type 876
ResultSet interface 870, 876, 878
absolute method 878
close method 871
column name 871
column number 871
CONCUR_READ_ONLY constant 877
CONCUR_UPDATABLE constant 877
concurrency constant 877
getInt method 871
getObject method 871, 878
getRow method 878
last method 878
next method 871
TYPE_FORWARD_ONLY constant 876
TYPE_SCROLL_INSENSITIVE constant 876
ResultSet Interface (cont.)
TYPE_SCROLL_SENSITIVE constant 877
ResultSetMetaData interface 870, 878
getColumnClassName method 878
getColumnCount method 870, 878
columnName method 878
columnType method 871
ResultSetTableModel enables a JTable to display the contents of a ResultSet 872
resumption model of exception handling 299
rethrow an exception 307
Return key 650, 651
return keyword 49, 122, 1025
return message in the UML 353
return type 49
in the UML 343, 348
of a method 41, 49
reusability 629
reusable software components 2, 123, 226
reuse 3, 31
reverse method of class
StringBuilder 520
reverse method of Collections 590, 596
reverseOrder method of Collections 592
RGB value 472, 473, 478
right aligned 454
right brace, { 25, 32, 71, 76
RIGHT constant of class FlowLayout 457
right justification 1053, 1062
right justify output 94
right justifying integers 1062
rigid area of class Box 725
.rmi file extension 685, 688
robust 33
robust application 293
role in the UML 330
role name in the UML 330
roll back a transaction 905
rollback method of interface Connection 905
rolling two dice 133
rollover Icon 420
root directory 542
root element (XML) 663
root html element 915
rotate method of class Graphics2D 501
round a floating-point number for display purposes 78
round-robin scheduling 740
rounded rectangle 486, 498
rounded rectangle (for representing a state in a UML state diagram) 338
rounding 1053
rounding a number 34, 73, 96, 116
RoundRectangle2D class 469
RoundRectangle2D. Double class 494, 498
row 851, 853, 856, 857, 858, 862
rowClasses attribute of h: dataTable 954
RowFilter class 884
rows of a two-dimensional array 167
rows to be retrieved 856
RowSet interface 885
Rule of Entity Integrity 855
Rule of Referential Integrity 853
rule of thumb (heuristic) 107
rules of operator precedence 34
run an applet in a web browser 656
run debugger command 1039
run method of interface Runnable 741, 840
Runnable interface 290, 741, 841
run method 741, 840
runnable state 738
running an application 12
running state 739
runtime error 11
runtime logic error 33
RuntimeException class 302
S
SaaS (Software as a Service) xxiii, 19
Salesforce 15
sandbox security model 660
SansSerif Java font 480
satisfaction 478
Saverin, Eduardo 17
saving account 93
scalar 159
scaling (random numbers) 126
scaling an image 673
scaling factor (random numbers) 126, 129
Scanner class 31, 32
hasNext method 101
next method 45
nextDouble method 59
nextLine method 44
scheduling threads 740
scientific notation 1055
scope 90
scope of a declaration 135
scope of a type parameter 631
scope of a variable 90
Screen class (ATM case study) 329, 330, 342, 349, 350, 351, 353, 355, 362
screen cursor 29
screen-manager program 257
scroll 431, 435
scroll arrow 432
scroll box 432
SCROLL_TAB_LAYOUT constant of class JTabbedPane 725
scrollbar 435, 466
of a JComboBox 432
scrollbar policies 466
SDK (Software Development Kit) 19
search engine 912
Second Life 15
secondary storage devices 540
sector 489
security 9
security certificate 660
SecurityException class 551
seed value (random numbers) 125, 129
SELECT SQL keyword 856, 857, 858, 859, 860
selected text in a JTextArea 466
selecting an item from a menu 410
selecting data from a table 852
selection 64
selection criteria 857
selection mode 435
selection statement 62, 63
if 63, 64, 65, 98
else 64, 65, 76, 98
switch 64, 98, 104
self target frame 807
Selvadurai, Naveen 18
semicolon (;) 25, 32, 38
send a message to an object 4
send data to a server 825
send message 52
set method of class DatagramSocket 829
sentence-style capitalization 402
sentinel-controlled repetition 74, 76
sentinel value 74, 76
separator character 545
separator line in a menu 706, 707
sequence 64, 582
sequence diagram in the UML 326, 351
sequence of messages in the UML 352
sequence structure 62
sequence-structure activity diagram 63
SequenceInputStream class 573
sequential-access file 540, 546, 812
sequential execution 62
sequential horizontal orientation in GroupLayout 1071
sequential layout of GUI components 1071
Serializable interface 290, 563
serialized object 562
Slerf Java font 480
server 802
server farm 935
server port number 825
server response 912
server-side artifacts 968
server-side form handler 911
server waits for connections from clients 811
server’s Internet address 813
server-side form handler 911
ServerSocket class 811, 818, 840
accept method 811, 818
service description for a web service 972
service endpoint interface (SEI) 968, 973
service of a class 192
ServiceManager class 672
lookup method 672
serviceName attribute of @WebService annotation 968
session 934
session expire (JSF) 935
session tracking 934
in web services 987
@SessionScoped annotation 934, 935, 937
set a value 51
set debugger command 1041
Set interface 580, 581, 605, 606, 608
set method of interface ListIterator 588
set method 51, 196
set of constants as an interface 279
SET SQL clause 863
set up event handling 413
setAlignment method of class FlowLayout 457
setAutoCommit method of interface Connection 905
setBackground method of class Component 435, 478
setBounds method of class Component 453
setCharAt method of class String 520
setColor method of class Graphics 473, 498
setCommand method of JdbcRowSet interface 887
setConstraints method of class GridBagLayout 731
setDefaultCloseOperation method of class JFrame 410, 699
setDisabledTextColor method of class JComponent 452
setEditable method of class JComponent 413
setError method of class System 541
setFileSelectionMode method of class JFileChooser 577
setFixedCellHeight method of class JList 437
setFixedCellWidth method of class JList 437
setFont method of class Component 425
setFont method of class Graphics 480
setForeground method of class JComponent 708
setHint method of class Manager 689
setHorizontalAlignment method of class JLabel 409
setHorizontalAlignmentBarPolicy method of class JScrollPane 467
setHorizontalAlignmentTextPosition method of class JLabel 409
setIcon method of class JLabel 409
setIn method of class System 541
setInverted method of class JSlider 696
setJMenuBar method of class JFrame 700, 707
setLayout method of class Container 408, 454, 460, 462, 725
setLineWrap method of class JTextArea 466
setListData method of class JList 438
setLocation method of class Component 453, 700
setLookAndFeel method of class UIManager 715
setMajorTickSpacing method of class JSlider 699
setMaximunRowCount method of class JButton 432
setMnemonic method of class AbstractButton 706
setOpaque method of class JComponent 446, 449
setOut method of System 541
setPage method of class JEditorPane 810
solid circle surrounded by a hollow circle in the UML 63
solid diamonds (representing composition) in the UML 330
sort 181
sort method
  of class Arrays 180
  of class Collections 591
SortDemo applet 649
sorted order 606, 608
SortedMap interface 608
SortedSet interface 606, 608
first method 608
last method 608
sorting
descending order 591
with a Comparator 592
sorting techniques 649
sound 656, 668
sound card 685
sound engine 685
sounds 692
source code 8, 251
Source view in Netbeans 1073
SourceForge 5
SOUTH constant of class BorderLayout 441, 457
SOUTH constant of class GridBagConstraints 726
SOUTHEAST constant of class GridBag-
Constraints 726
SOUTHWEST constant of class GridBag-
Constraints 726
space character 24
space flag 1065
spacing between components in GroupLayout 1072
speaker 685
special character 32, 503
specialization 225
specialization in the UML 366
specifics 257
.sql file extension 686
-splash command-line option to the java command 1083
splash screen 1083
SplashScreen class 1084
split method of class String 529, 535
SpreadSheet applet 649
SQL 850, 852, 855, 856, 862
DELETE statement 856, 864
FROM clause 856
GROUP BY 856
IDENTITY keyword 890
INNER JOIN clause 856, 861
INSERT statement 856, 862
LIKE clause 858
ON clause 861
ORDER BY clause 856, 859, 860
SELECT query 856, 857, 858, 859, 860
SET clause 863
UPDATE statement 856
VALUES clause 862
WHERE clause 857
.sql 866
SQL (Structured Query Language) 889
SQL keyword 855
SQL script 866
SQL statement 905
SQLException class 870, 871, 890
SQLFeatureNotSupportedException class 877
sqlrt method of class Math 116, 117, 122
square brackets, [] 142
square root 117
stack 628
Stack class 604
isEmpty method 604
of package java.util 602
peek method 604
pop method 604
push method 603
Stack generic class 629
Stack< Double > 636
Stack< Integer > 636
Stack generic class declaration 629
stack trace 295
stack unwinding 308
StackTraceElement class 311
getClassName method 311
getFileName method 311
getLineNumber method 311
getMethodName method 311
state value 749
standard error stream 298, 307, 1053
standard error stream (System.err) 541, 571
standard input stream (System.in) 32, 541
standard output stream 307
standard output stream (System.out) 25, 541, 571
standard reusable component 226
standard time format 190
start method of class JApplet 654, 657, 659
start method of class Timer 679
start method of interface Player 690
start tag 663
starting angle 488
startsWith method of class String 510
starvation 740
state 326
state button 423
state dependent 753
state diagram for the ATM object 338
state diagram in the UML 338
state in the UML 326, 339
state machine diagram in the UML 326, 338
state of an object 333, 338
stateChanged method of interface
ChangeListener 699
stateless protocol 934
statement 25, 41
Statement interface 870, 871, 889
close method 871
executeQuery method 870
Statements
break 102, 105, 106
continue 105
control statement 62, 64, 65
control-statement nesting 64
control-statement stacking 64
do...while 64, 96, 97
double selection 64
empty 38, 68
empty statement 68
enhanced for 157
Statements (cont.)
for 64, 89, 91, 92, 93, 95
if 35, 63, 64, 65, 98
if...else 64, 65, 76, 98
looping 64
multiple selection 64
nested if...else 66, 67
repetition 62, 64, 68
return 122
selection 62, 63
single selection 64
switch 64, 98, 104
switch multiple-selection statement 129
try 153
while 64, 68, 69, 72, 76, 87

statements
throw 190
try-with-resources 316

static
class member 210
class variable 211
field (class variable) 210
import 213
import on demand 214
keyword 116, 1025
method 41, 95
static binding 278
static initializer block 937
status bar 653
step debugger command 1043
step up debugger
command 1044

Stone, Isaac “Biz” 17
stop debugger command 1039
stop method
of JApplet 654, 657
stop method of class Timer 680
stop method of interface AudioClip 685
store method of Properties 614
stored procedure 904
stream 307, 1053
stream header 819
stream of bytes 540
stream socket 802, 813, 833
stream-based communications 802
streams 802
streams-based transmission 825
strictfp keyword 1025
string 25
of characters 25
String class 503
charAt method 505, 520
compareTo method 507, 509
concat method 514
endsWith method 510
equals method 507, 509
equalsIgnoreCase method 507, 509
format method 190, 1070
getChars method 505
immutable 212
indexOf method 511
lastIndexOf method 511
length method 505
matches method 530
regionMatches method 507
replaceAll method 535
replaceAll method 535
split method 529, 535
startsWith method 510
substring method 513
toCharArray method 516
toLowerCase method 588
toLowerCase method 515
toUpperCase method 588
toUpperCase method 515
trim method 516
valueOf method 516
StringBuilder class (cont.)
getChars method 520
insert method 523
length method 518
reverse method 520
setCharAt method 520
StringBuilder class (cont.)
StringIndexOutOfBoundsException class 513, 520
StringReader class 573
Strings in switch statements 105
StringWriter class 573, 980
Stroke object 497, 498
strongly typed language 85
Stroustrup, Bjarne 293
structure 915
structure of a system 337, 338
structured programming 62
Structured Query Language (SQL) 850, 852, 855
style attribute of h:panelGrid 925
styleClass attribute of a JSF element 931
styleClass attribute of h:dataTable 954
subclass 225, 365, 366
subdirectory 649
sublist 588
subList method of List 588
submenu 701
submit method of class ExecutorService 799
subprotocol for communication 869
substring (index) 142
substring method of class String 513
subtraction 34
operator, -
subtraction compound assignment operator, -= 82
suffix F for float literals 604
suffix L for long literals 603
sum the elements of an array 147
summarizing responses to a survey 150
Sun Audio file format (.au extension) 685, 688
super keyword 228, 250, 1025
call superclass constructor 242
superclass 225, 365, 366
constructor 232
constructor call syntax 242
default constructor 232
direct 225, 226
indirect 225, 226
method overridden in a subclass 250
suspend an applet's execution 657
sweep 488
sweep counterclockwise 488
.swf file extension 688
Swing Event Package 125
Swing GUI APIs 399
Swing GUI components 399
Swing GUI components package 125
swing.properties file xxxi, 401
SwingConstants interface 290, 409, 699
SwingSet3 demo 399
SwingUtilities class 716, 818
invokeLater method 818
updateComponent-TreeUI method 716
SwingWorker class 785
cancel method 799
doInBackground method 785, 788
done method 785, 788
execute method 785
get method 785
isCancelled method 794
process method 786, 795
publish method 785, 795
setProgress method 786, 795
switch logic 105
switch multiple-selection statement 64, 98, 104, 129, 1025
activity diagram with break statements 104
case label 102
comparing Strings 105
controlling expression 102
default case 102, 104, 129
Sybase 850
synchronization 744, 764
synchronization wrapper 615
synchronize 737
synchronize access to a collection 582
synchronized keyword 615, 745, 1025
method 745
statement 745
synchronized collection 582
synchronous call 352
synchronous error 301
synchronous request 956
SynchronousQueue class 784
syntax error 26
system 325
system behavior 325
System class
arraycopy 181, 182
exit method 304, 551
setErr method 541
setIn method 541
setOut method 541
system requirements 323
system service 811
system structure 325
System.err (standard error stream) 298, 541, 571, 1053
System.in (standard input stream) 541
System.out
print method 27, 28
printf method 29
println method 25, 28
System.out (standard output stream) 25, 541, 571
SystemColor class 497
SystemTray class 1087
addTrayIcon method 1088
defaultTrayIcon method 1088
removeTrayIcon method 1088
T
tab 1068
tab character, \t 29
Tab key 25
tab stops 25, 29
table 167, 851
table element 167
table of values 167
truncToSize method of
   class ArrayList<T> 184
two-dimensional array 73
two-dimensional graphics 324
two-dimensional shapes 469
type 32
type argument 631
type casting 77
type-import-on-demand declaration 219
type inference with the « notation (Java SE 7) 585
type parameter 623, 629, 636
   scope 631
   section 623, 629
type variable 623
type-wraper class 524,
   580, 625
   implements Comparable 625
也为FORWARD ONLY constant 876
TYPE_INT_RGB constant of
   class BufferedImage 498
TYPE_SCROLL_INSENSITIVE constant 876
TYPE_SCROLL_SENSITIVE constant 877
Types class 871
typesetting system 503
typing in a text field 410

U
UDP (User Datagram Protocol) 802, 826
ui : repeat element 941
UIManager class 715
   getInstalledLookAndFeel method 715
LookAndFeelInfo nested class 715
   setLookAndFeel method 715
UIManager .LookAndFeelInfo class
   getClassName method 715
UML (Unified Modeling Language) 5, 319, 325,
   329, 336, 337, 365
   activity diagram 62, 63,
   65, 69, 91, 97
   aggregation 331
   arrow 63
   association 329
   class diagram 43
   compartment in a class diagram 43
   diagram 325
   diamond 64
dotted line 63
   elided diagram 329
   final state 63
   frame 354
   guard condition 65
   guilemet (« and ») 55
   hollow diamond representing aggregation 331
   many-to-one relationship 332
   merge symbol 69
   multiplicity 329
   note 63
   one-to-many relationship 332
   one-to-one relationship 332
   Resource Center
   (www.deitel.com/ UML/) 326
   role name 330
   solid circle 63
   solid circle surrounded by a hollow circle 63
   solid diamond representing composition 330
   Specification 331
   UML (www.uml.org) 63
   UML Activity Diagram
   small diamond symbol
   (for representing a decision) in the UML 341
   solid circle (for representing an initial state) in the UML 339
   solid circle enclosed in an open circle (for representing the end of an activity) in the UML 339
   UML Class Diagram 329
   attribute compartment 336
   operation compartment 342
   UML Sequence Diagram
   activation 353
   arrowhead 353
   lifeline 353
   UML State Diagram
   rounded rectangle (for representing a state) in the UML 338
   solid circle (for representing an initial state) in the UML 339
   UML Use Case Diagram
   acotr 324
   use case 325
   unary operator 78, 110
   cast 77
   unboxing 629, 634
   unboxing conversion 581
   uncaught exception 299
   unchecked exceptions 302
   uncovering a component 472
   underlying data structure 604
   underscore (_) SQL wildcard character 857, 858
   uneditable JTextArea 464
   uneditable text or icons 404
   unknown value of the character typed 857
   Unicode character set 85
   Unicode value of the character typed 85
   URL (Universal Resource Locator) 542, 803, 909
   universal-time format 188, 189
   UNIX 8. 25, 101, 551, 648
   UnknownHostException class 813
   unlock method of
   interface Lock 776, 781
   unmarshal method of class
   JAXB 983
   unmodifiable collection 582
   unmodifiable wrapper 615
   unspecified number of arguments 177
   UnsupportedOperationException class 588
   unwatch debugger command 1048
   unwinding the method-call stack 308
   UPDATE SQL statement 856, 863
   updateComponentTreeUI method of class SwingUtilities 716
   upper bound 625
   of a wildcard 642
   upper bound of a type parameter 626, 627
   upper-left corner (0, 0)
   coordinates of an applet 653
   upper-left corner of a GUI component 469
   upper-left x-coordinate 473
   upper-left y-coordinate 473
   uppercase letter 24, 32
   URI (Uniform Resource Identifier) 542, 803
   URI class
   toURL method 692
   URL (Uniform Resource Locator) 542, 803, 804, 909
   URL class 685
   openStream method 987
   use case diagram in the UML 324, 325
   use case in the UML 324
   use case modeling 324
   User Datagram Protocol (UDP) 802, 826
   user interface 913
   Utilities Package 124
   utility method 104
v option of the jar command 662
va 552
valid identifier 32
validate method of class Container 462
validation 926
validatorMessage attribute of a JSF element 931
Validators (JSF)
  f:validateBean 926
  f:validateDoubleRange 926
  f:validateLength 926
  f:validateLongRange 926
  f:validateRegex 926
  f:validateRequired 926
validity checking 203
value of a param 804
value of an attribute 915
valueChanged method of interface ListSelectionLister 435
valueOf method of class String 516
values method of an enum 208
VALUES SQL clause 862
var attribute of h:dataTable 954
var attribute of var:repeat 941
variable 30, 32
  name 32
  reference type 52
  variable declaration statement 32
variable is not modifiable 215
variable-length argument list 177
variable scope 90
Vector class 186, 582
vendor element of a JNLP document 664
verb phrase in requirements document 342
VERTICAL constant of class GridBagConstraints 726
vertical coordinate 469
vertical gap space 460
vertical scrolling 466
vertical strut 724
VERTICAL_SCROLLBAR_ALWAYS constant of class JScrollPane 467
VERTICAL_SCROLLBAR_AS_NEEDED constant of class JScrollPane 467
VERTICAL_SCROLLBAR_NEVER constant of class JScrollPane 467
vi 8
video 668, 692
video game 126
video sharing 15
View 400
view 588
view (in MVC) 922
view a shape from different angles 649
virtual directory 910
virtual key code 452
virtual machine (VM) 8
virtual world 15
visibility in the UML 360
visibility marker in the UML 360
visual feedback 423
void keyword 25, 41, 1025
VoIP (Voice over IP) 15
volatile keyword 1025
W
WADL (Web Application Description Language) 981
wait for a new connection 818
wait method of class
  Object 253, 763
  waiting line 581, 604
  waiting state 738
  waiting thread 766
  watch debugger command 1046
  waterfall model 323
  .wav file extension 685
  web 803
  Web 2.0 15
  web app development 908
  web application
    Ajax 957
    traditional 956
  Web Application
    Description Language (WADL) 981
  web application framework 908
  Web Application project 966
  web browser 647, 807
  execute an applet 652, 656
Web Form 935
web server 811, 909
Web Service Description Language (WSDL) 972
web service host 965
web service reference 974
web services 16, 963
  adding a web service reference to an application 973
  client-side artifacts 973
  consuming a web service 965
  deploying a web service 970
@GET annotation 980
GlassFish application server's Tester web page 971
implemented as a class 965
JAX-RS 963
JAX-WS 963
  name attribute of @WebService annotation 968
@Path annotation 979
@PathParam annotation 980
POJO (Plain Old Java Object) 916, 968
  processing user-defined types 1009
@Produces annotation 980
proxy class 968, 973
  publishing a web service 965, 970
RequestContext interface 1001
REST 963
server-side artifacts 968
  serviceName attribute of @WebService annotation 968
session tracking 990
SOAP 973
  test a web service 971
  testing a web service from another computer 972
  web service host 965
  web service reference 974
web services (cont.)
  @WebMethod annotation 968
  @WebParam annotation 969
  @WebService annotation 968
  @WebMethod annotation 968
  operationName attribute 968
  @WebParam annotation 969
  name attribute 969
  @WebService annotation 968
  name attribute 968
  serviceName attribute 968
weightx field of class GridBagConstraints 727
weighty field of class GridBagConstraints 727
WEST constant of class BorderLayout 441, 457
WEST constant of class GridBagConstraints 726
WHERE SQL clause 856, 857, 858, 860, 863, 864
while repetition statement 64, 68, 69, 72, 76, 87, 1025
activity diagram in the UML 69
white space 24, 25
white-space character 516, 529, 530
whole/part relationship 330
widgets 399
width 484
width attribute (CSS) 925
width attribute of the applet-desc element 664
width of a rectangle in pixels 473
width of an applet in pixels 655
Wikipedia 15
wildcard 642
  in a generic type parameter 640
  type argument 642
  upper bound 642
Williams, Evan 17
window 699
Window class 699
addWindowListener method 700
dispose method 699
pack method 719
window event 700
window event-handling methods 443
window events 700
window gadgets 399
windowActivated method of interface
WindowListener 700
WindowAdapter class 443, 884
windowClosing method of interface
WindowListener 700, 884
windowClosing method of interface
WindowListener 700
WindowConstants interface 699
DISPOSE_ON_CLOSE constant 699
DO NOTHING_ON_CLOSE constant 699
HIDE_ON_CLOSE constant 699
windowDeactivated method of interface
WindowListener 700
windowDeiconified method of interface
WindowListener 700
windowIconified method of interface
WindowListener 700
windowOpened method 700
windowOpened method of interface Window-
Listener 700
Windows 8, 101, 551, 648
Windows look-and-feel 695
Windows Performance Package 688
Windows Wave file format (.wav extension) 685
WireFrame applet 649
Withdrawal class (ATM case study) 329, 330,
331, 334, 340, 341, 342, 350, 351, 353, 354, 355,
362, 363, 365, 366, 367, 370
write boolean method of interface DataOutput
writeByte method of interface DataOutput
writeBytes method of interface DataOutput
writeChar method of interface DataOutput
writeChars method of interface DataOutput
writeDouble method of interface DataOutput
writeFloat method of interface DataOutput
writeInt method of interface DataOutput
writeLong method of interface DataOutput
writeObject method of class ObjectOutputStream
of interface ObjectOutput
Writer class 573, 573
writeShort method of interface DataOutput
writeUTF method of interface DataOutput
WSDL (Web Service Description Language) 972
XHTML 1.0 Strict Recommendation 915
Transitional Recommendation 915
XML (eXtensible Markup Language) 663, 908,
972
declaration 915
element 683
empty element 925
end tag 663
root element 663
start tag 663
vocabulary 663
XMLHttpRequest object 956
xmlns attributes 915
y-coordinate 469, 493
Y_AXIS constant of class Box 725
y-axis 469
YouTube 15, 18
Z
0 (zero) flag 1064, 1066
zero-based counting 144
zeroth element 142
Zuckerberg, Mark 17
x-coordinate 469, 493, 653
X_AXIS constant of class Box 725
x-axis 469
XHTML (eXtensible HyperText Markup Language) 908, 909, 915
applet element 655
body element 655
document 654
page 909
tag 654