Contents in Brief

Chapter 1 **Introduction to Computers and Java** 1

Chapter 2 **Java Fundamentals** 27

Chapter 3 **Decision Structures** 111

Chapter 4 **Loops and Files** 189

Chapter 5 **Methods** 269

Chapter 6 **A First Look at Classes** 317

Chapter 7 **Arrays and the `ArrayList` Class** 403

Chapter 8 **A Second Look at Classes and Objects** 493

Chapter 9 **Text Processing and More about Wrapper Classes** 557

Chapter 10 **Inheritance** 611

Chapter 11 **Exceptions and Advanced File I/O** 701

Chapter 12 **JavaFX: GUI Programming and Basic Controls** 759

Chapter 13 **JavaFX: Advanced Controls** 823

Chapter 14 **JavaFX: Graphics, Effects, and Media** 909

Chapter 15 **Recursion** 999

Chapter 16 **Sorting, Searching, and Algorithm Analysis** 1027

Chapter 17 **Generics** 1079

Chapter 18 **Collections and the Stream API** 1125

Chapter 19 **Linked Lists** 1195

Chapter 20 **Stacks and Queues** 1245

Chapter 21 **Binary Trees, AVL Trees, and Priority Queues** 1287

Index 1353

Appendices A–M Companion Website

Chapters 22–25 Companion Website

Case Studies 1–7 Companion Website

Contents

Preface xxv

Chapter 1 **Introduction to Computers and Java 1**

1.1 Introduction 1
1.2 Why Program? 1
1.3 Computer Systems: Hardware and Software 2
Hardware *2*
Software *5*
1.4 Programming Languages 6
What Is a Program? *6*
A History of Java *8*
1.5 What Is a Program Made Of? 8
Language Elements *8*
Lines and Statements *11*
Variables *11*
The Compiler and the Java Virtual Machine *12*
Java Software Editions *13*
Compiling and Running a Java Program *14*
1.6 The Programming Process 16
Software Engineering *18*
1.7 Object-Oriented Programming 19
Review Questions and Exercises 21
Programming Challenge 25

Chapter 2 **Java Fundamentals 27**

2.1 The Parts of a Java Program 27
2.2 The `print` and `println` Methods, and the Java API 33
2.3 Variables and Literals 39
Displaying Multiple Items with the + Operator *40*
Be Careful with Quotation Marks *41*
More about Literals *42*
Identifiers *42*
Class Names *44*

2.4 Primitive Data Types 44
The Integer Data Types 46
Floating-Point Data Types 47
The `boolean` Data Type 50
The `char` Data Type 50
Variable Assignment and Initialization 52
Variables Hold Only One Value at a Time 53
2.5 Arithmetic Operators 54
Integer Division 57
Operator Precedence 57
Grouping with Parentheses 59
The `Math` Class 62
2.6 Combined Assignment Operators 63
2.7 Conversion between Primitive Data Types. 65
Mixed Integer Operations 67
Other Mixed Mathematical Expressions. 68
2.8 Creating Named Constants with `final` 69
2.9 The `String` Class 70
Objects Are Created from Classes. 71
The `String` Class 71
Primitive Type Variables and Class Type Variables 71
Creating a `String` Object. 72
2.10 Scope 76
2.11 Comments 78
2.12 Programming Style 83
2.13 Reading Keyboard Input. 85
Reading a Character 89
Mixing Calls to `nextLine` with Calls to Other `Scanner` Methods 89
2.14 Dialog Boxes 93
Displaying Message Dialogs 93
Displaying Input Dialogs. 94
An Example Program. 94
Converting String Input to Numbers 96
2.15 Common Errors to Avoid 99
Review Questions and Exercises 100
Programming Challenges 106

Chapter 3 **Decision Structures 111**

3.1 The `if` Statement 111
Using Relational Operators to Form Conditions. 113
Putting It All Together 114
Programming Style and the `if` Statement 117
Be Careful with Semicolons 117
Having Multiple Conditionally Executed Statements 118
Flags 118
Comparing Characters. 119

3.2 The `if-else` Statement 121
3.3 Nested `if` Statements 124
3.4 The `if-else-if` Statement 128
3.5 Logical Operators 134
The Precedence of Logical Operators 139
Checking Numeric Ranges with Logical Operators. 140
3.6 Comparing String Objects 142
Ignoring Case in String Comparisons. 146
3.7 More about Variable Declaration and Scope 147
3.8 The Conditional Operator (Optional) 149
3.9 The switch Statement 150
3.10 Displaying Formatted Output with `System.out.printf` and `String.format` 160
Format Specifier Syntax 163
Precision 164
Specifying a Minimum Field Width 164
Flags 167
Formatting String Arguments. 170
The `String.format` Method. 172
3.11 Common Errors to Avoid 174
Review Questions and Exercises 175
Programming Challenges 181

Chapter 4 **Loops and Files 189**

4.1 The Increment and Decrement Operators 189
The Difference between Postfix and Prefix Modes 192
4.2 The `while` Loop 193
The `while` Loop Is a Pretest Loop 196
Infinite Loops. 196
Don’t Forget the Braces with a Block of Statements 197
Programming Style and the `while` Loop 198
4.3 Using the `while` Loop for Input Validation 200
4.4 The `do-while` Loop 204
4.5 The `for` Loop. 207
The `for` Loop Is a Pretest Loop 210
Avoid Modifying the Control Variable in the Body of the `for` Loop 211
Other Forms of the Update Expression. 211
Declaring a Variable in the `for` Loop’s Initialization Expression 211
Creating a User Controlled `for` Loop. 212
Using Multiple Statements in the Initialization and Update Expressions 213
4.6 Running Totals and Sentinel Values 216
Using a Sentinel Value 219
4.7 Nested Loops 221
4.8 The break and continue Statements (Optional) 229
4.9 Deciding Which Loop to Use 229

4.10 Introduction to File Input and Output 230
Using the `PrintWriter` Class to Write Data to a File 230
Appending Data to a File 236
Specifying the File Location 237
Reading Data from a File 237
Reading Lines from a File with the `nextLine` Method 238
Adding a `throws` Clause to the Method Header 241
Checking for a File's Existence 245
4.11 Generating Random Numbers with the `Random` Class 249
4.12 Common Errors to Avoid 255
Review Questions and Exercises 256
Programming Challenges 262

Chapter 5 **Methods 269**

5.1 Introduction to Methods 269
`void` Methods and Value-Returning Methods 270
Defining a `void` Method 271
Calling a Method 272
Layered Method Calls 276
Using Documentation Comments with Methods 277
5.2 Passing Arguments to a Method 279
Argument and Parameter Data Type Compatibility 281
Parameter Variable Scope 282
Passing Multiple Arguments 282
Arguments Are Passed by Value 284
Passing Object References to a Method 285
Using the `@param` Tag in Documentation Comments 288
5.3 More about Local Variables *290*
Local Variable Lifetime 292
Initializing Local Variables with Parameter Values 292
5.4 Returning a Value from a Method 293
Defining a Value-Returning Method 293
Calling a Value-Returning Method 294
Using the `@return` Tag in Documentation Comments 296
Returning a `boolean` Value 300
Returning a Reference to an Object 300
5.5 Problem Solving with Methods 302
Calling Methods That Throw Exceptions 305
5.6 Common Errors to Avoid 305
Review Questions and Exercises 306
Programming Challenges 311

Chapter 6 **A First Look at Classes 317**

6.1 Objects and Classes 317
Classes: Where Objects Come From 318

Classes in the Java API 319
Primitive Variables vs. Objects 321
6.2 Writing a Simple Class, Step by Step 324
Accessor and Mutator Methods 338
The Importance of Data Hiding 338
Avoiding Stale Data 339
Showing Access Specification in UML Diagrams 339
Data Type and Parameter Notation in UML Diagrams 339
Layout of Class Members. 340
6.3 Instance Fields and Methods 341
6.4 Constructors 346
Showing Constructors in a UML Diagram 348
Uninitialized Local Reference Variables 348
The Default Constructor. 348
Writing Your Own No-Arg Constructor. 349
The `String` Class Constructor 350
6.5 Passing Objects as Arguments 358
6.6 Overloading Methods and Constructors 370
The `BankAccount` Class 372
Overloaded Methods Make Classes More Useful 378
6.7 Scope of Instance Fields 378
Shadowing 379
6.8 Packages and `import` Statements 380
Explicit and Wildcard `import` Statements 380
The `java.lang` Package 381
Other API Packages 381
6.9 Focus on Object-Oriented Design: Finding the Classes and Their Responsibilities 382
Finding the Classes 382
Identifying a Class's Responsibilities 385
This Is Only the Beginning. 388
6.10 Common Errors to Avoid 388
Review Questions and Exercises 389
Programming Challenges 394

Chapter 7 **Arrays and the `ArrayList` Class 403**

7.1 Introduction to Arrays. 403
Accessing Array Elements. 405
Inputting and Outputting Array Contents 406
Java Performs Bounds Checking 409
Watch Out for Off-by-One Errors 410
Array Initialization. 411
Alternate Array Declaration Notation 412
7.2 Processing Array Elements 413
Array Length 415
The Enhanced `for` Loop 416

Letting the User Specify an Array's Size *417*
Reassigning Array Reference Variables. *419*
Copying Arrays *420*
7.3 Passing Arrays as Arguments to Methods 422
7.4 Some Useful Array Algorithms and Operations
Comparing Arrays 426
Summing the Values in a Numeric Array *427*
Getting the Average of the Values in a Numeric Array *428*
Finding the Highest and Lowest Values in a Numeric Array. *428*
The `SalesData` Class. *429*
Partially Filled Arrays *437*
Working with Arrays and Files *438*
7.5 Returning Arrays from Methods 439
7.6 `String` Arrays. 441
Calling `String` Methods from an Array Element. *443*
7.7 Arrays of Objects. 444
7.8 The Sequential Search Algorithm. 447
7.9 Two-Dimensional Arrays. 450
Initializing a Two-Dimensional Array. *454*
The `length` Field in a Two-Dimensional Array *455*
Displaying All the Elements of a Two-Dimensional Array. *457*
Summing All the Elements of a Two-Dimensional Array. *457*
Summing the Rows of a Two-Dimensional Array *458*
Summing the Columns of a Two-Dimensional Array *458*
Passing Two-Dimensional Arrays to Methods *459*
Ragged Arrays *461*
7.10 Arrays with Three or More Dimensions 462
7.11 The Selection Sort and the Binary Search Algorithms 463
The Selection Sort Algorithm *463*
The Binary Search Algorithm *466*
7.12 Command-Line Arguments and Variable-Length Argument Lists 468
Command-Line Arguments *469*
Variable-Length Argument Lists. *470*
7.13 The `ArrayList` Class 472
Creating and Using an `ArrayList` Object. *473*
Using the Enhanced `for` Loop with an `ArrayList`. *474*
The `ArrayList` Class's `toString` method *475*
Removing an Item from an `ArrayList`. *476*
Inserting an Item. *477*
Replacing an Item *478*
Capacity *479*
Storing Your Own Objects in an `ArrayList` *479*
Using the Diamond Operator for Type Inference *480*
7.14 Common Errors to Avoid 481
Review Questions and Exercises 481
Programming Challenges 486

Chapter 8 A Second Look at Classes and Objects 493

8.1 Static Class Members 493

A Quick Review of Instance Fields and Instance Methods 493

Static Members 494

Static Fields 494

Static Methods. 497

8.2 Passing Objects as Arguments to Methods. 500

8.3 Returning Objects from Methods 503

8.4 The `toString` Method 505

8.5 Writing an `equals` Method. 509

8.6 Methods That Copy Objects. 512

Copy Constructors 514

8.7 Aggregation. 515

Aggregation in UML Diagrams 523

Security Issues with Aggregate Classes 523

Avoid Using `null` References 525

8.8 The `this` Reference Variable 528

Using `this` to Overcome Shadowing. 529

Using `this` to Call an Overloaded Constructor from Another Constructor. 530

8.9 Enumerated Types 531

Enumerated Types Are Specialized Classes 532

Switching On an Enumerated Type 538

8.10 Garbage Collection. 540

The `finalize` Method 542

8.11 Focus on Object-Oriented Design: Class Collaboration 542

Determining Class Collaborations with CRC Cards. 545

8.12 Common Errors to Avoid 546

Review Questions and Exercises 547

Programming Challenges 551

Chapter 9 Text Processing and More about Wrapper Classes 557

9.1 Introduction to Wrapper Classes 557

9.2 Character Testing and Conversion with the `Character` Class. 558

Character Case Conversion 563

9.3 More `String` Methods. 566

Searching for Substrings 566

Extracting Substrings. 572

Methods That Return a Modified `String` 576

The Static `valueOf` Methods 577

9.4 The `StringBuilder` Class. 579

The `StringBuilder` Constructors 580

Other `StringBuilder` Methods 581

The toString Method 584
9.5 Tokenizing Strings 589
9.6 Wrapper Classes for the Numeric Data Types 594
The Static toString Methods 594
The toBinaryString, toHexString, and toOctalString Methods . . . 594
The MIN_VALUE and MAX_VALUE Constants 595
Autoboxing and Unboxing. 595
9.7 Focus on Problem Solving: The TestScoreReader Class 597
9.8 Common Errors to Avoid 601
Review Questions and Exercises 601
Programming Challenges 605

Chapter 10 **Inheritance 611**

10.1 What Is Inheritance? 611
Generalization and Specialization 611
Inheritance and the "Is a" Relationship 612
Inheritance in UML Diagrams 620
The Superclass's Constructor 621
Inheritance Does Not Work in Reverse 623
10.2 Calling the Superclass Constructor 624
When the Superclass Has No Default or No-Arg Constructors 630
Summary of Constructor Issues in Inheritance. 631
10.3 Overriding Superclass Methods 632
Overloading versus Overriding. 637
Preventing a Method from Being Overridden. 640
10.4 Protected Members 641
Package Access 646
10.5 Chains of Inheritance 647
Class Hierarchies 653
10.6 The Object Class 653
10.7 Polymorphism. 655
Polymorphism and Dynamic Binding 656
The "Is-a" Relationship Does Not Work in Reverse 658
The instanceof Operator. 659
10.8 Abstract Classes and Abstract Methods 660
Abstract Classes in UML. 666
10.9 Interfaces. 667
An Interface is a Contract 669
Fields in Interfaces 673
Implementing Multiple Interfaces 673
Interfaces in UML. 673
Default Methods 674
Polymorphism and Interfaces 676
10.10 Anonymous Inner Classes. 681
10.11 Functional Interfaces and Lambda Expressions 684

10.12 Common Errors to Avoid .689
Review Questions and Exercises 690
Programming Challenges 696

Chapter 11 **Exceptions and Advanced File I/O 701**

11.1 Handling Exceptions .701
Exception Classes 702
Handling an Exception 703
Retrieving the Default Error Message 707
Polymorphic References to Exceptions 710
Using Multiple `catch` Clauses to Handle Multiple Exceptions 710
The `finally` Clause 718
The Stack Trace 720
Handling Multiple Exceptions with One `catch` Clause 721
When an Exception Is Not Caught 723
Checked and Unchecked Exceptions 724
11.2 Throwing Exceptions .725
Creating Your Own Exception Classes 728
Using the `@exception` Tag in Documentation Comments 731
11.3 Advanced Topics: Binary Files, Random Access Files, and Object Serialization .732
Binary Files 732
Random Access Files 739
Object Serialization 744
Serializing Aggregate Objects 748
11.4 Common Errors to Avoid .749
Review Questions and Exercises 749
Programming Challenges 755

Chapter 12 **JavaFX: GUI Programming and Basic Controls 759**

12.1 Graphical User Interfaces .759
Event-Driven GUI Programs 761
12.2 Introduction to JavaFX .762
Controls 762
Stages and Scenes 763
The `Application` Class 763
12.3 Creating Scenes .765
Creating Controls 766
Creating Layout Containers 766
Creating a `Scene` Object 767
Adding the `Scene` Object to the Stage 768
Setting the Size of the Scene 770
Aligning Controls in an `HBox` Layout Container 770
12.4 Displaying Images .772

Loading Images from an Internet Location 775
Setting the Size of an Image 775
Preserving the Image's Aspect Ratio 775
Changing an ImageView's Image 776
12.5 More about the `HBox`, `VBox`, and `GridPane` Layout Containers 776
The `HBox` Layout Container 777
The `VBox` Layout Container 782
The `GridPane` Layout Container 784
Using Multiple Layout Containers in the Same Screen 791
12.6 `Button` Controls and Events 792
Handling Events 794
Writing Event Handlers 794
Registering an Event Handler 795
12.7 Reading Input with `TextField` Controls 799
12.8 Using Anonymous Inner Classes and Lambda Expressions to Handle Events 803
Using Anonymous Inner Classes to Create Event Handlers 803
Using Lambda Expressions to Create Event Handlers 806
12.9 The `BorderPane` Layout Container 808
12.10 The `ObservableList` Interface 812
12.11 Common Errors to Avoid 814
Review Questions and Exercises 814
Programming Challenges 818

Chapter 13 **JavaFX: Advanced Controls 823**

13.1 Styling JavaFX Applications with CSS 823
Type Selector Names 824
Style Properties 825
Applying a Stylesheet to a JavaFX Application 826
Applying Styles to the Root Node 830
Specifying Multiple Selectors in the Same Style Definition 832
Working with Colors 832
Creating a Custom Style Class Name 834
ID Selectors 836
Inline Style Rules 837
13.2 `RadioButton` Controls 838
Determining in Code Whether a `RadioButton` Is Selected 839
Selecting a `RadioButton` in Code 839
Responding to `RadioButton` Clicks 844
13.3 `CheckBox` Controls 848
Determining in Code Whether a `CheckBox` Is Selected 848
Selecting a `CheckBox` in Code 849
Responding to `CheckBox` Clicks 853
13.4 `ListView` Controls 853
Retrieving the Selected Item 855

Retrieving the Index of the Selected Item . . . 856
Responding to Item Selection with an Event Handler. . . 859
Adding Items versus Setting Items . . . 860
Initializing a `ListView` with an Array or an `ArrayList`. . . 861
Selection Modes. . . 862
Retrieving Multiple Selected Items . . . 863
Working With the Elements of an ObservableList. . . 866
Converting an `ObservableList` to an Array . . . 867
Using Code to Select an Item in a `ListView`. . . 868
`ListView` Orientation . . . 868
Creating ListViews of Objects Other Than `String` . . . 869
13.5 `ComboBox` Controls . . . 874
Retrieving the Selected Item. . . 875
Responding to `ComboBox` Item Selection with an Event Handler . . . 877
Editable ComboBoxes . . . 879
13.6 `Slider` Controls. . . 880
13.7 `TextArea` Controls . . . 885
13.8 Menus . . . 887
Assigning Mnemonics to Menu Items . . . 895
13.9 The `FileChooser` Class . . . 897
Displaying a `FileChooser` Dialog Box . . . 898
13.10 Using Console Output to Debug a GUI Application . . . 898
13.11 Common Errors to Avoid . . . 902
Review Questions and Exercises 902
Programming Challenges 906

Chapter 14 **JavaFX: Graphics, Effects, and Media 909**

14.1 Drawing Shapes . . . 909
The Screen Coordinate System . . . 909
The `Shape` Class and Its Subclasses . . . 910
The `Line` Class. . . 911
Changing the Stroke Color. . . 914
The `Circle` Class. . . 914
The `Rectangle` Class. . . 918
The `Ellipse` Class. . . 921
The `Arc` Class. . . 924
The `Polygon` Class. . . 928
The `Polyline` Class. . . 931
The `Text` Class. . . 933
Rotating Nodes . . . 936
Scaling Nodes . . . 938
14.2 Animation . . . 940
The `TranslateTransition` Class. . . 941
The `RotateTransition` Class. . . 944
The `ScaleTransition` Class. . . 949

The StrokeTransition Class . . . 952
The FillTransition Class . . . 953
The FadeTransition Class . . . 954
Controlling the Animation . . . 956
Specifying an Interpolator . . . 956
14.3 Effects . . . 958
The DropShadow Class . . . 958
The InnerShadow Class . . . 961
The ColorAdjust Class . . . 962
The BoxBlur, GaussianBlur, and MotionBlur Classes . . . 962
The SepiaTone Class . . . 965
The Glow Class . . . 966
The Reflection Class . . . 967
Combining Effects . . . 967
14.4 Playing Sound Files . . . 969
Registering an EndOfMedia Event Handler . . . 971
14.5 Playing Videos . . . 974
14.6 Handling Key Events . . . 979
Using an Anonymous Inner Class to Register a Key Event Handler to the Scene . . . 980
Using a Lambda Expression to Register a Key Event Handler to the Scene . . . 981
14.7 Handling Mouse Events . . . 986
14.8 Common Errors to Avoid . . . 992
Review Questions and Exercises 992
Programming Challenges 995

Chapter 15 **Recursion** 999

15.1 Introduction to Recursion . . . 999
15.2 Solving Problems with Recursion . . . 1002
Direct and Indirect Recursion . . . 1006
15.3 Examples of Recursive Methods . . . 1007
Summing a Range of Array Elements with Recursion . . . 1007
Drawing Concentric Circles . . . 1008
The Fibonacci Series . . . 1011
Finding the Greatest Common Divisor . . . 1012
15.4 A Recursive Binary Search Method . . . 1014
15.5 The Towers of Hanoi . . . 1017
15.6 Common Errors to Avoid . . . 1021
Review Questions and Exercises 1022
Programming Challenges 1025

Chapter 16 **Sorting, Searching, and Algorithm Analysis** 1027

16.1 Introduction to Sorting Algorithms . . . 1027
The Bubble Sort . . . 1027

Using the Bubble Sort to Sort Objects *1033*
The Selection Sort. *1036*
Using the Selection Sort to Sort Objects *1040*
The Insertion Sort. *1042*
Using the Insertion Sort to Sort Objects *1047*
The Quicksort Algorithm *1048*
Using Quicksort to Sort Objects *1052*
16.2 Introduction to Search Algorithms 1052
The Sequential Search Algorithm *1052*
Average Case Analysis of the Sequential Search *1055*
The Binary Search *1055*
Efficiency of the Binary Search *1059*
A Recursive Binary Search *1059*
16.3 Analysis of Algorithms 1063
Computational Problems and Basic Steps *1064*
Complexity of Algorithms *1065*
Worst Case Complexity of Algorithms *1066*
Average Case Complexity. *1067*
Asymptotic Complexity and the Big O Notation *1068*
16.4 Common Errors to Avoid 1072
Review Questions and Exercises 1073
Programming Challenges 1076

Chapter 17 **Generics 1079**

17.1 Introduction to Generics 1079
Using the Diamond Operator for Type Inference *1082*
17.2 Writing a Generic Class 1082
Only Reference Types Can Be Passed to Type Parameters *1087*
Instantiating a Generic Class without Specifying a Type Argument . . . *1088*
Commonly Used Type Parameter Names *1090*
17.3 Passing Objects of a Generic Class to a Method 1090
Constraining a Type Parameter *1092*
Defining a Type Parameter in a Method Header *1094*
The `extends` *Key Word Constrains a Type to an Upper Bound* *1095*
The `super` *Key Word Constrains a Type to a Lower Bound* *1095*
17.4 Writing Generic Methods 1096
17.5 Constraining a Type Parameter in a Generic Class 1097
17.6 Inheritance and Generic Classes 1100
17.7 Defining Multiple Type Parameters 1104
17.8 Generics and Interfaces 1107
Constraining a Type Parameter to a Type That Implements an Interface *1108*
17.9 Erasure 1113
17.10 Restrictions on the Use of Generic Types 1116
17.11 Common Errors to Avoid 1118
Review Questions and Exercises 1119
Programming Challenges 1122

Chapter 18 **Collections and the Stream API 1125**

18.1 Introduction to the Java Collections Framework 1125
Lists, Sets, and Maps 1125
Java Functional Interfaces 1126
JCF Interfaces 1126
Iterating a Collection 1127
The Iterable Interface and the forEach method 1127
Iterators 1128
The Enhanced For Loop 1131
The Collection Interface 1131
18.2 Lists 1133
The List Interface 1134
The ArrayList and LinkedList Classes 1135
Creating and Initializing Lists 1136
List Iterators 1137
Methods of the LinkedList Class 1140
Using an Interface Variable to Reference a Collection Object 1141
18.3 Sets 1143
The Set Interface 1143
The HashSet Class 1143
Implementing Your Own hashCode Method 1148
The LinkedHashSet Class 1152
The SortedSet Interface and the TreeSet Class 1153
Using a Comparator to Compare Objects 1156
18.4 Maps 1158
The HashMap Class 1160
The LinkedHashMap Class 1166
The SortedMap Interface and the TreeMap Class 1168
18.5 The Collections Class 1169
18.6 The Stream API 1171
The Stream Package 1172
Creating a Stream 1172
Terminal Operations 1173
Intermediate Operations on Streams 1177
Intermediate Stream Operations are Lazy 1179
The Stream map Operation 1181
The Stream reduce Methods 1182
Stream Collectors 1184
18.7 Common Errors to Avoid 1189
Review Questions and Exercises 1190
Programming Challenges 1193

Chapter 19 **Linked Lists 1195**

19.1 Introduction to Linked Lists 1195
Creating Lists and Adding Nodes 1196

Removing a Node *1198*
Traversing Linked Lists *1199*
19.2 Operations on Linked Lists 1202
The `isEmpty` Method *1202*
The `size` Method *1203*
The `add` Methods *1203*
The `remove` Methods *1204*
The `toString` Method *1204*
A Graphical Interface for the Linked List Program *1210*
19.3 Doubly Linked and Circularly Linked Lists 1215
Adding a Node to a Doubly Linked List *1216*
Removing a Node from a Doubly Linked List *1217*
Circularly Linked Lists. *1223*
19.4 Recursion on Linked Lists 1224
A Recursive Implementation of `size` *1224*
A Recursive Implementation of the `add(String e, Node list)`
Method *1225*
A Recursive Implementation of `add(int index, String e, Node list)` . .*1226*
A Recursive Implementation of the `remove` Methods *1226*
19.5 Common Errors to Avoid 1238
Review Questions and Exercises 1238
Programming Challenges 1241

Chapter 20 **Stacks and Queues 1245**

20.1 Stacks and Their Applications 1245
Examples and Applications of Stacks *1245*
Stack Operations *1246*
Stacks of Primitive Types *1247*
20.2 Array Implementation of Stacks 1248
The Stack Push Operation *1249*
The Stack `empty` Method *1249*
The Stack `peek` and `pop` Methods *1249*
Stacks of Objects *1253*
20.3 Linked Implementation of Stacks 1254
Implementation of Stack Methods. *1255*
20.4 Queues and Their Applications 1259
20.5 Array Implementation of Queues 1259
GUI Front End for the Queue Demo Program. *1266*
20.6 Linked List Implementation of Queues 1270
Queue Initialization and Enqueuing of Items *1270*
Dequeuing Items *1271*
20.7 Generic Implementation of Stacks and Queues 1275
20.8 Queues and Breadth-First Search 1278
20.9 Common Errors to Avoid 1281
Review Questions and Exercises 1282
Programming Challenges 1284

Chapter 21 **Binary Trees, AVL Trees, and Priority Queues 1287**

21.1 Binary Trees and Their Applications 1287
Binary Tree Concepts 1288
Applications of Binary Trees 1289
Representing Nodes of Binary Trees 1289
Traversing a Binary Tree 1290
Graphical Display of Binary Trees 1293
Class Implementation of Binary Trees 1297
21.2 Binary Search Trees 1297
Adding a Value to a Binary Search Tree 1299
Removing a Value from a Binary Search Tree 1299
Implementation of Binary Search Trees 1303
Graphical Display of Binary Search Trees 1308
A Program for Testing the Binary Search Tree Class 1310
21.3 AVL Trees 1315
Adding New Elements to AVL Trees 1316
Implementation of AVL Trees 1319
21.4 Priority Queues 1328
Applications of Priority Queues 1330
Using Comparators with Priority Queues 1331
Analysis of Heapsort 1333
Implementation of Priority Queues 1333
Binary Trees with the Heap Order Property 1333
Complete Binary Trees and Heaps 1334
The Depth of Complete Binary Trees 1335
Storing a Complete Binary Tree in an Array 1335
Adding an Item to a Heap 1336
Removing the Minimum Element. 1337
21.5 Common Errors to Avoid 1345
Review Questions and Exercises 1345
Programming Challenges 1349

Index 1353

The following appendices, online chapters, and online case studies are available on the book's online resource page at www.pearson.com/gaddis.

Online Appendices

Appendix A: The ASCII/Unicode Characters
Appendix B: Operator Precedence and Associativity
Appendix C: Java Key Words
Appendix D: Installing the JDK and Using the JDK Documentation
Appendix E: Using the `javadoc` Utility
Appendix F: More about the Math Class
Appendix G: Packages
Appendix H: Working with Records and Random Access Files
Appendix I: Configuring JavaDB and Installing Apache Derby
Appendix J: The QuickSort Algorithm
Appendix K: Named Colors
Appendix L: Answers to Checkpoints
Appendix M: Answers to Odd-Numbered Review Questions

Online Chapters

Chapter 22: Databases
Chapter 23: A First Look at GUI Applications with Swing
Chapter 24: Advanced Swing GUI Applications
Chapter 25: Applets and More

Online Case Studies

Case Study 1: Calculating Sales Commission
Case Study 2: The `Amortization` Class
Case Study 3: The `PinTester` Class
Case Study 4: Parallel Arrays
Case Study 5: The `FeetInches` Class
Case Study 6: The `SerialNumber` Class
Case Study 7: A Simple Text Editor Application

LOCATION OF VIDEONOTES IN THE TEXT

Chapter 1	Compiling and Running a Java Program, p. 14 Using an IDE, p. 14 Your First Java Program, p. 25
Chapter 2	Displaying Console Output, p. 33 Declaring Variables, p. 39 Simple Math Expressions, p. 55 The Miles-per-Gallon Problem, p. 107
Chapter 3	The `if` Statement, p. 111 The `if-else` Statement, p. 120 The `if-else-if` Statement, p. 129 The Time Calculator Problem, p. 182
Chapter 4	The while Loop, p. 193 The Pennies for Pay Problem, p. 263
Chapter 5	Passing Arguments to a Method, p. 279 Returning a Value from a Method, p. 293 The Retail Price Calculator Problem, p. 311
Chapter 6	Writing Classes and Creating Objects, p. 325 Initializing an Object with a Constructor, p. 346 The Personal Information Class Problem, p. 395
Chapter 7	Accessing Array Elements in a Loop, p. 407 Passing an Array to a Method, p. 422 The Charge Account Validation Problem, p. 487
Chapter 8	Returning Objects from Methods, p. 503 Aggregation, p. 515 The BankAccount , Class Copy Constructor Problem, p. 552
Chapter 9	The Sentence Capitalizer Problem, p. 605
Chapter 10	Inheritance, p. 611 Polymorphism, p. 655 The `Employee` and `Productionworker` Classes Problem, p. 696
Chapter 11	Handling Exceptions, p. 701 The Exception Project Problem, p. 757
Chapter 12	Introduction to JavaFX, p. 762 Creating Scenes, p. 765 Displaying Images, p. 772 The HBox Layout Container, p. 777 The VBox Layout Container, p. 782 The `GridPane` Layout Container, p. 784 `Button` Controls and Events, p. 792 The TextField Control, p. 799 Using Anonymous Inner Classes as Event Handlers, p. 803 Using Lambda Expressions as Event Handlers, p. 806 The Latin Translator Problem, p. 818

(*continued on the next page*)

LOCATION OF VIDEONOTES IN THE TEXT *(continued)*

Chapter 13	JavaFX and CSS, p. 823 `RadioButton` Controls, p. 838 `CheckBox` Controls, p. 848 `ListView` Controls, p. 853 `ComboBox` Controls, p. 874 `Slider` Controls, p. 880 The Dorm and Meal Plan Calculator Problem, p. 906
Chapter 14	Drawing Shapes with JavaFX, p. 909 JavaFX Animation, p. 940 JavaFX Effects, p. 958 Playing Sound Files with JavaFX, p. 970 Playing Videos with JavaFX, p. 974 Handling Key Events in JavaFX, p. 979 Handling Mouse Events in JavaFX, p. 986 The Coin Toss Problem, p. 996
Chapter 15	Reducing a Problem with Recursion, p. 1003 The Recursive Power Problem, p. 1026
Chapter 16	The Bubble Sort Algorithm, p. 1027 Searching for Objects with the Binary Search Algorithm, p. 1076
Chapter 17	Inheritance and Generic classes, p. 1100 Highest and Lowest Elements, p. 1123
Chapter 18	The `HashMap` Class, p. 1160
Chapter 19	Adding a Node to a Doubly-Linked List, p. 1216 Recursive Linked List Operations, p. 1242
Chapter 20	Array Implementation of Queues, p. 1259 Array-Based Deque, p. 1285
Chapter 21	Removing a Value from a Binary Search Tree, p. 1299 Prefix Calculator, p. 1350

Preface

Welcome to *Starting Out with Java: From Control Structures through Data Structures, Fourth Edition*. This book is intended for a traditional two-semester CS1/CS2 sequence of courses. The first half of the book, intended for a CS1 class, teaches fundamental programming and problem-solving concepts using Java. The second half of the book, intended for a CS2 class, teaches advanced concepts and provides an introduction to algorithms and data structures. The book is written for students with no prior programming background, but experienced students will also benefit from its depth of detail.

Control Structures First, Then Objects, Then Data Structures

The text introduces students to the fundamentals of data types, input/output, control structures, methods, and objects created from standard library classes. Next, students learn to use arrays. Then, a series of more advanced topics are presented, including inheritance, polymorphism, GUI development, recursion, searching, sorting, algorithm analysis, and generics. Finally, students are introduced to the world of data structures, beginning with an overview of the Java Collections Framework (JCF), and continuing through a series of chapters in which students learn to develop their own data structures, including linked lists, stacks, queues, priority queues, binary trees, and AVL trees. Students are shown how to implement data structures either with or without generics, which gives instructors a great deal of flexibility in presenting the material.

As with all the books in the *Starting Out With* series, the hallmark of the text is its clear, friendly, and easy-to-understand writing. In addition, it is rich in concise, practical example programs.

Changes in This Edition

This book's pedagogy, organization, and clear writing style remain the same as in the previous edition. The most significant change in this edition is the switch from Swing to JavaFX in the chapters that focus on GUI development. Although Swing is not officially deprecated, Oracle has announced that JavaFX has replaced Swing as the standard GUI library for Java[1].

[1] http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html#6

In this edition, we have added the following new chapters:

- **Chapter 12 JavaFX: GUI Programming and Basic Controls:** This chapter presents the basics of developing graphical user interface (GUI) applications with JavaFX. Fundamental controls, layout containers, and the basic concepts of event-driven programming are covered.
- **Chapter 13 JavaFX: Advanced Controls:** This chapter discusses CSS styling and advanced user interface controls.
- **Chapter 14 JavaFX: Graphics, Effects, and Media:** This chapter discusses 2D shapes, animation, visual effects, playing audio and video, and responding to mouse and keyboard events.

The Swing and Applet material that appeared in the previous edition is still available on the book's companion Web site, as the following online chapters:

- The previous Chapter 12 *A First Look At GUI Applications* is now available online as Chapter 23.
- The previous Chapter 13 *Advanced GUI Applications* is now available online as Chapter 24.
- The previous Chapter 14 *Applets and More* is now available online as Chapter 25.

In addition to the new JavaFX chapters, the online Database chapter, which is now Chapter 22, has been updated to use JavaFX instead of Swing for its GUI applications. We have also added several new, motivational programming problems throughout the book.

Organization of the Text

The text teaches Java step by step. Each chapter covers a major set of topics and builds knowledge as students progress through the book. Although the chapters can be easily taught in their existing sequence, there is some flexibility. Figure P-1 shows chapter dependencies. Each box represents a chapter or a group of chapters. An arrow points from a chapter to the chapter that must be previously covered.

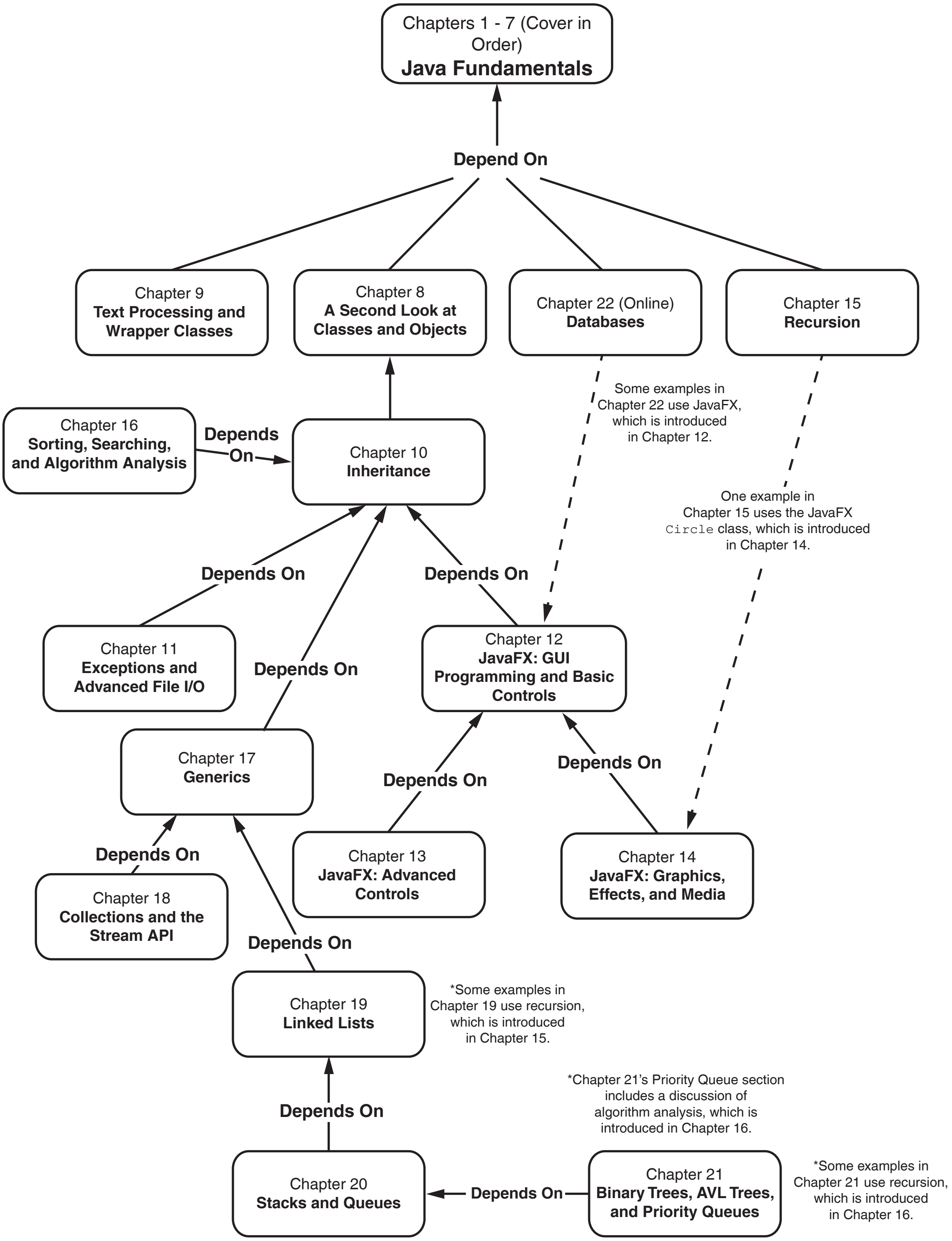
Brief Overview of Each Chapter

Chapter 1: Introduction to Computers and Java. This chapter provides an introduction to the field of computer science, and covers the fundamentals of hardware, software, and programming languages. The elements of a program, such as key words, variables, operators, and punctuation are discussed by examining a simple program. An overview of entering source code, compiling, and executing a program is presented. A brief history of Java is also given.

Chapter 2: Java Fundamentals. This chapter gets students started in Java by introducing data types, identifiers, variable declarations, constants, comments, program output, and simple arithmetic operations. The conventions of programming style are also introduced. Students learn to read console input with the Scanner class and with dialog boxes using `JOptionPane`.

Chapter 3: Decision Structures. In this chapter students explore relational operators and relational expressions, and are shown how to control the flow of a program with the `if`,

Figure P-1 Chapter dependencies


Chapters 1 - 7 (Cover in Order)
Java Fundamentals
Depend On
Chapter 9
Text Processing and Wrapper Classes
Chapter 8
A Second Look at Classes and Objects
Chapter 22 (Online)
Databases
Chapter 15
Recursion
Chapter 16
Sorting, Searching, and Algorithm Analysis
Depends On
Chapter 10
Inheritance
Some examples in Chapter 22 use JavaFX, which is introduced in Chapter 12.
One example in Chapter 15 uses the JavaFX Circle class, which is introduced in Chapter 14.
Depends On
Depends On
Chapter 11
Exceptions and Advanced File I/O
Depends On
Chapter 12
JavaFX: GUI Programming and Basic Controls
Chapter 17
Generics
Depends On
Depends On
Chapter 13
JavaFX: Advanced Controls
Chapter 14
JavaFX: Graphics, Effects, and Media
Depends On
Chapter 18
Collections and the Stream API
Depends On
Chapter 19
Linked Lists
*Some examples in Chapter 19 use recursion, which is introduced in Chapter 15.
Depends On
*Chapter 21's Priority Queue section includes a discussion of algorithm analysis, which is introduced in Chapter 16.
Chapter 20
Stacks and Queues
Depends On
Chapter 21
Binary Trees, AVL Trees, and Priority Queues
*Some examples in Chapter 21 use recursion, which is introduced in Chapter 16.

if-else, and if-else-if statements. Nested if statements, logical operators, the conditional operator, and the switch statement are also covered. The chapter discusses how to compare String objects with the equals, compareTo, equalsIgnoreCase, and compareToIgnoreCase methods. Formatting numeric output with the System.out.printf method and the String.format method is discussed.

Chapter 4: Loops and Files. This chapter covers Java's repetition control structures. The while loop, do-while loop, and for loop are taught, along with common uses for these devices. Counters, accumulators, running totals, sentinels, and other application-related topics are discussed. Simple file operations for reading and writing text files are included.

Chapter 5: Methods. In this chapter students learn how to write void methods, value-returning methods, and methods that do and do not accept arguments. The concept of functional decomposition is discussed.

Chapter 6: A First Look at Classes. This chapter introduces students to designing classes for the purpose of instantiating objects. Students learn about class fields and methods, and UML diagrams are introduced as a design tool. Then constructors and overloading are discussed. A BankAccount class is presented as a case study, and a section on object-oriented design is included. This section leads the students through the process of identifying classes and their responsibilities within a problem domain. There is also a section that briefly explains packages and the import statement.

Chapter 7: Arrays and the ArrayList Class. In this chapter students learn to create and work with single and multi-dimensional arrays. Numerous array-processing techniques are demonstrated, such as summing the elements in an array, finding the highest and lowest values, and sequentially searching an array are also discussed. Other topics, including ragged arrays and variable-length arguments (varargs) are also discussed. The ArrayList class is introduced and Java's generic types are briefly discussed and demonstrated.

Chapter 8: A Second Look at Classes and Objects. This chapter shows students how to write classes with added capabilities. Static methods and fields, interaction between objects, passing objects as arguments, and returning objects from methods are discussed. Aggregation and the "has a" relationship is covered, as well as enumerated types. A section on object-oriented design shows how to use CRC cards to determine the collaborations among classes.

Chapter 9: Text Processing and More about Wrapper Classes. This chapter discusses the numeric and Character wrapper classes. Methods for converting numbers to strings, testing the case of characters, and converting the case of characters are covered. Autoboxing and unboxing are also discussed. More String class methods are covered, including using the split method to tokenize strings. The chapter also covers the StringBuilder class.

Chapter 10: Inheritance. The study of classes continues in this chapter with the subjects of inheritance and polymorphism. The topics covered include superclasses, subclasses, how constructors work in inheritance, method overriding, polymorphism and dynamic binding, protected and package access, class hierarchies, abstract classes, abstract methods, anonymous inner classes, interfaces, and lambda expressions.

Chapter 11: Exceptions and Advanced File I/O. In this chapter students learn to develop enhanced error trapping techniques using exceptions. Handling exceptions is covered, as well as developing and throwing custom exceptions. The chapter discusses advanced techniques for working with sequential access, random access, text, and binary files.

Chapter 12: JavaFX: GUI Programming and Basic Controls. This chapter presents the basics of developing graphical user interface (GUI) applications with JavaFX. Fundamental controls, layout containers, and the basic concepts of event-driven programming are covered.

Chapter 13: JavaFX: Advanced Controls. This chapter discusses CSS styling and advanced user interface controls, such as RadioButtons, CheckBoxes, ListViews, ComboBoxes, Sliders, and TextAreas. Menu systems and FileChoosers are also covered.

Chapter 14: JavaFX: Graphics, Effects, and Media. This chapter discusses 2D shapes, animation, visual effects, playing audio and video, and responding to mouse and keyboard events.

Chapter 15: Recursion. This chapter presents recursion as a problem-solving technique. Numerous examples of recursive methods are demonstrated.

Chapter 16: Sorting, Searching, and Algorithm Analysis. In this chapter students learn the basics of sorting arrays and searching for data stored in them. The chapter covers the bubble sort, selection sort, insertion sort, Quicksort, sequential search, and binary search algorithms. A section on algorithm analysis is also provided, which covers basic steps, complexity, and Big O notation.

Chapter 17: Generics. This chapter shows students how to write generic classes and methods. Topics include erasure, passing instances of a generic class as arguments to a method, constraining type parameters, generics and inheritance, and generics and interfaces. Restrictions on the use of generic types are also discussed.

Chapter 18: Collections and the Stream API. This chapter introduces students to the Java Collections Framework (JCF). Lists, sets, maps, and the Collections class are discussed. The chapter concludes with a discussion of the classes and interfaces in the java.util.stream package.

Chapter 19: Linked Lists. This chapter introduces concepts and techniques for writing a linked list class. Basic linked list operations, such as adding, inserting, removing, and traversing, are covered. Doubly-linked lists, circularly-linked lists, and using recursion with linked lists are discussed.

Chapter 20: Stacks and Queues. In this chapter students learn about the operations of stacks and queues, and how to write array-based and linked list-based stack and queue classes.

Chapter 21: Binary Trees, AVL Trees, and Priority Queues. This chapter covers various aspects of binary trees and binary tree operations. The specific binary tree implementations covered are binary search trees, AVL trees, and priority queues.

Chapter 22 (Online): Databases. This chapter is available on the book's companion Web site (at www.pearson.com/gaddis). It introduces the students to database programming. The basic concepts of database management systems and SQL are first introduced. Then the students learn to use JDBC to write database applications in Java. Relational data is covered, and numerous example programs are presented throughout the chapter.

Chapters 23 – 25 (Online): Legacy Swing GUI Chapters. The previous edition's chapters on Swing GUI and Applet programming are now available on the book's companion website (at www.pearson.com/gaddis) as Chapters 23 through 25.

Features of the Text

Concept Statements. Each major section of the text starts with a concept statement that concisely summarizes the focus of the section.

Example Programs. The text has an abundant number of complete and partial example programs, each designed to highlight the current topic. In most cases the programs are practical, real-world examples.

Program Output. Each example program is followed by a sample of its output, which shows students how the program functions.

Checkpoints. Checkpoints, highlighted by the checkmark icon, appear at intervals throughout each chapter. They are designed to check students' knowledge soon after learning a new topic. Answers for all Checkpoint questions can be downloaded from the book's companion Web site at www.pearson.com/gaddis.

> **NOTE:** Notes appear at several places throughout the text. They are short explanationsof interesting or often misunderstood points relevant to the topic at hand.

> **TIP:** Tips advise the student on the best techniques for approaching different programming problems and appear regularly throughout the text.

> **WARNING!** Warnings caution students about certain Java features, programming techniques, or practices that can lead to malfunctioning programs or lost data.

In the Spotlight. Many of the chapters provide an *In the Spotlight* section that presents a programming problem, along with detailed, step-by-step analysis showing the student how to solve it.

VideoNote

VideoNotes. A series of online videos, developed specifically for this book, are available for viewing at www.pearson.com/gaddis. Icons appear throughout the text, alerting the student to videos about specific topics.

Case Studies. Case studies that simulate real-world business applications are introduced throughout the text and are available for download from the Gaddis resource page at www.pearson.com/gaddis.

Common Errors to Avoid. Most chapters provide a list of common errors and explanations of how to avoid them.

Review Questions and Exercises. Each chapter presents a thorough and diverse set of review questions and exercises. They include Multiple Choice and True/False, Find the Error, Algorithm Workbench, and Short Answer.

Programming Challenges. Each chapter offers a pool of programming challenges designed to solidify students' knowledge of topics at hand. In most cases the assignments present real-world problems to be solved.

Supplements

Student Online Resources

Many student resources are available for this book from the publisher. The following items are available on the Gaddis Series resource page at www.pearson.com/gaddis:

- The source code for each example program in the book
- Access to the book's companion VideoNotes
- Appendixes A–M (listed in the Contents)
- A collection of seven valuable Case Studies (listed in the Contents)
- Links to download the Java™ Edition Development Kit
- Links to download numerous programming environments, including jGRASP™, Eclipse™, TextPad™, NetBeans™, JCreator, and DrJava

Instructor Resources

The following supplements are available to qualified instructors:

- Answers to all of the Review Questions
- Solutions for the Programming Challenges
- PowerPoint presentation slides for each chapter

Acknowledgments

There have been many helping hands in the development and publication of this book. We would like to thank the following faculty reviewers for their helpful suggestions and expertise:

John Bono
George Mason University

Irene Bruno
George Mason University

Richard Burns
Westchester University

Jackie Horton
University of Vermont

Mohammad T. Islam
Southern Connecticut State University

David Krebs
University of Pittsburgh

Michael Ruth
Roosevelt University

Christian Servin
El Paso Community College

Li Yang
Western Michigan University

Reviewers for Previous Editions

Ahmad Abuhejleh
University of Wisconsin, River Falls

Colin Archibald
Valencia Community College

Ijaz Awani
Savannah State University

Bill Bane
Tarleton State University

N. Dwight Barnette
Virginia Tech

Asoke Bhattacharyya
Saint Xavier University, Chicago

Marvin Bishop
Manhattan College

eather Booth
University of Tennessee, Knoxville

David Boyd
Valdosta State University

Julius Brandstatter
Golden Gate University

Kim Cannon
Greenville Tech

Jesse Cecil
College of the Siskiyous

James Chegwidden
Tarrant County College

Kay Chen
Bucks County Community College

Brad Chilton
Tarleton State University

Diane Christie
University of Wisconsin, Stout

Cara Cocking
Marquette University

Jose Cordova
University of Louisiana, Monroe

Walter C. Daugherity
Texas A&M University

Michael Doherty
University of the Pacific

Jeanne M. Douglas
University of Vermont

Sander Eller
California Polytechnic University, Pomona

Brooke Estabrook-Fishinghawk
Mesa Community College

Mike Fry
Lebanon Valley College

David Goldschmidt
College of St. Rose

Georgia R. Grant
College of San Mateo

Carl Stephen Guynes
University of North Texas

Nancy Harris
James Madison University

Chris Haynes
Indiana University

Ric Heishman
Northern Virginia Community College

Deedee Herrera
Dodge City Community College

Mary Hovik
Lehigh Carbon Community College

Brian Howard
DePauw University

Alan Jackson
Oakland Community College (MI)

Norm Jacobson
University of California, Irvine

Zhen Jiang
West Chester University

Stephen Judd
University of Pennsylvania

Neven Jurkovic
Palo Alto College

Dennis Lang
Kansas State University

Jiang Li
Austin Peay State University

Harry Lichtbach
Evergreen Valley College

Michael A. Long
California State University, Chico

Cheng Luo
Coppin State University

Tim Margush
University of Akron

Blayne E. Mayfield
Oklahoma State University

Scott McLeod
Riverside Community College

Dean Mellas
Cerritos College

Georges Merx
San Diego Mesa College

Martin Meyers
California State University, Sacramento

Pati Milligan
Baylor University

Laurie Murphy
Pacific Lutheran University

Steve Newberry
Tarleton State University

Lynne O'Hanlon
Los Angeles Pierce College

Merrill Parker
Chattanooga State Technical
Community College

Bryson R. Payne
North Georgia College and State
University

Rodney Pearson
Mississippi State University

Peter John Polito
Springfield College

Charles Robert Putnam
California State University, Northridge

Y. B. Reddy
Grambling State University

Elizabeth Riley
Macon State College

Felix Rodriguez
Naugatuck Valley Community College

Diane Rudolph
John A Logan College

Carolyn Schauble
Colorado State University

Bonnie Smith
Fresno City College

Daniel Spiegel
Kutztown University

Caroline St. Clair
North Central College

Karen Stanton
Los Medanos College

Peter van der Goes
Rose State College

Timothy Urness
Drake University

Tuan A Vo
Mt. San Antonio College

Xiaoying Wang
University of Mississippi

Yu Wu
University of North Texas

Zijiang Yang
Western Michigan University

The authors would like to thank their families for their tremendous support during the development of this book. We also want to thank everyone at Pearson for making the *Starting Out With* series so successful. We are extremely fortunate to have Matt Goldstein as our editor and Meghan Jacoby as editorial assistant. Their support and encouragement makes it a pleasure to write chapters and meet deadlines. We are also fortunate to have Demetrius Hall as our marketing manager. He consistently does a great job getting this book out to the academic community. We had a great production team, led by Amanda Brands. She worked tirelessly to make this edition a reality. Thanks to you all!

About the Authors

Tony Gaddis is the principal author of the *Starting Out With* series of textbooks. Tony has nearly 20 years experience teaching computer science courses at Haywood Community College in North Carolina. He is a highly acclaimed instructor who was previously selected as the North Carolina Community College Teacher of the Year and has received the Teaching Excellence award from the National Institute for Staff and Organizational Development. The *Starting Out With* series includes introductory books using the C++ programming language, the Java™ programming language, Microsoft® Visual Basic®, Microsoft® C#®, Python, Programming Logic and Design, MIT App Inventor, and Alice, all published by Pearson.

Godfrey Muganda is Professor of Computer Science at North Central College. He teaches a wide variety of courses at the undergraduate and graduate levels including courses in Linux and Unix programming, Windows and .NET programming, web application development, web services, data structures, and algorithms. He is a past winner of the North Central College faculty award for outstanding scholarship. His primary research interests are in the area of fuzzy sets and systems.

CHAPTER 1

Introduction to Computers and Java

TOPICS

1.1 Introduction
1.2 Why Program?
1.3 Computer Systems: Hardware and Software
1.4 Programming Languages
1.5 What Is a Program Made Of?
1.6 The Programming Process
1.7 Object-Oriented Programming

1.1 Introduction

This book teaches programming using Java. Java is a powerful language that runs on practically every type of computer. It can be used to create large applications, small programs, mobile applications, and code that powers a Web site. Before plunging right into learning Java, however, this chapter will review the fundamentals of computer hardware and software, and then take a broad look at computer programming in general.

1.2 Why Program?

CONCEPT: Computers can do many different jobs because they are programmable.

Every profession has tools that make the job easier to do. Carpenters use hammers, saws, and measuring tapes. Mechanics use wrenches, screwdrivers, and ratchets. Electronics technicians use probes, scopes, and meters. Some tools are unique and can be categorized as belonging to a single profession. For example, surgeons have certain tools that are designed specifically for surgical operations. Those tools probably aren't used by anyone other than surgeons. There are some tools, however, that are used in several professions. Screwdrivers, for instance, are used by mechanics, carpenters, and many others.

The computer is a tool used by so many professions that it cannot be easily categorized. It can perform so many different jobs that it is perhaps the most versatile tool ever made. To the accountant, computers balance books, analyze profits and losses, and prepare tax reports. To the factory worker, computers control manufacturing machines and track production. To the mechanic, computers analyze the various systems in an automobile and pinpoint hard-to-find problems. The computer can do such a wide variety of tasks because it can be *programmed*. It is a machine specifically designed to follow instructions. Because of the computer's programmability, it doesn't belong to any single profession. Computers are designed to do whatever job their programs, or *software*, tell them to do.

Computer programmers do a very important job. They create software that transforms computers into the specialized tools of many trades. Without programmers, the users of computers would have no software, and without software, computers would not be able to do anything.

Computer programming is both an art and a science. It is an art because every aspect of a program should be carefully designed. Here are a few of the things that must be designed for any real-world computer program:

- The logical flow of the instructions
- The mathematical procedures
- The layout of the programming statements
- The appearance of the screens
- The way information is presented to the user
- The program's "user friendliness"
- Manuals, help systems, and/or other forms of written documentation

There is also a science to programming. Because programs rarely work right the first time they are written, a lot of analyzing, experimenting, correcting, and redesigning is required. This demands patience and persistence of the programmer. Writing software demands discipline as well. Programmers must learn special languages such as Java because computers do not understand English or other human languages. Programming languages have strict rules that must be carefully followed.

Both the artistic and scientific nature of programming makes writing computer software like designing a car: Both cars and programs should be functional, efficient, powerful, easy to use, and pleasing to look at.

1.3 Computer Systems: Hardware and Software

CONCEPT: All computer systems consist of similar hardware devices and software components.

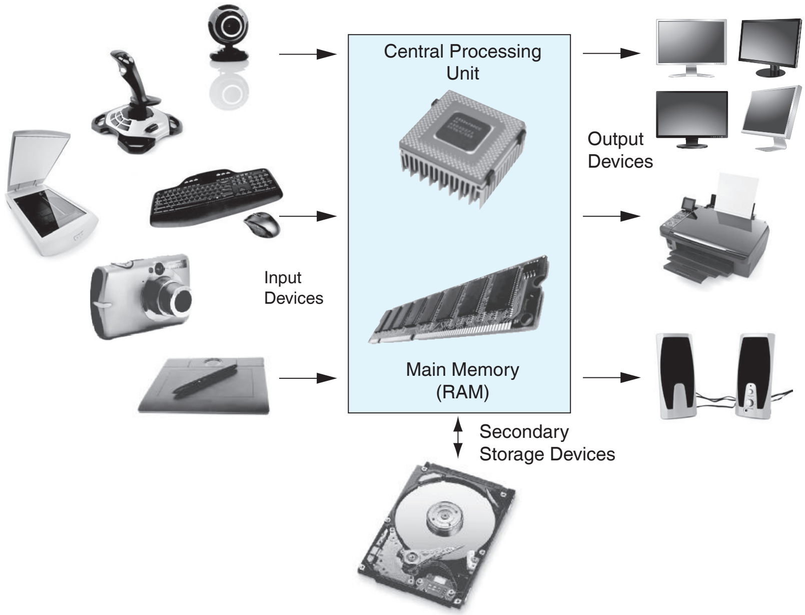
Hardware

Hardware refers to the physical components that a computer is made of. A computer, as we generally think of it, is not an individual device, but a system of devices. Like the instruments in a symphony orchestra, each device plays its own part. A typical computer system consists of the following major components:

- The central processing unit (CPU)
- Main memory
- Secondary storage devices
- Input devices
- Output devices

The organization of a computer system is shown in Figure 1-1.

Figure 1-1 The organization of a computer system 1-1a (TL) iko/Shutterstock, 1-1b (TL) Nikita Rogul/Shutterstock, 1-1c (TLC) Feng Yu/Shutterstock , 1-1d (TLC) Chiyacat/Shutterstock, 1-1e Elkostas/Shutterstock, 1-1f (TLB) tkemot/Shutterstock, 1-1g (BC) Vitaly Korovin/Shutterstock, 1-1h (TRB) Lusoimages/Shutterstock, 1-1i (TRC) jocic/Shutterstock, 1-1j (TR) Best Pictures here/Shutterstock, 1-1k (MC) Peter Guess/Shutterstock, 1-1l (TC) Aquila/Shutterstock

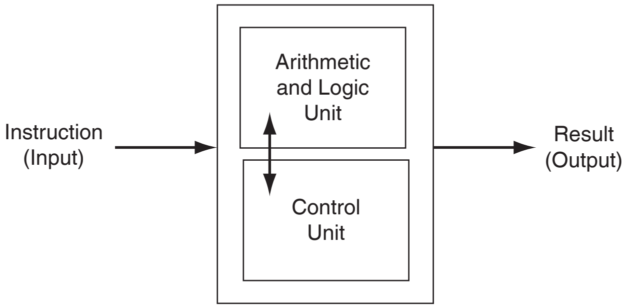


Let's take a closer look at each of these devices.

The CPU

At the heart of a computer is its *central processing unit*, or *CPU*. The CPU's job is to fetch instructions, follow the instructions, and produce some resulting data. Internally, the central processing unit consists of two parts: the *control unit* and the *arithmetic and logic unit (ALU)*. The control unit coordinates all of the computer's operations. It is responsible for determining where to get the next instruction and regulating the other major components of the computer with control signals. The arithmetic and logic unit, as its name suggests, is designed to perform mathematical operations. The organization of the CPU is shown in Figure 1-2.

Figure 1-2 The organization of the CPU



A program is a sequence of instructions stored in the computer's memory. When a computer is running a program, the CPU is engaged in a process known formally as the *fetch/decode/execute cycle*. The steps in the fetch/decode/execute cycle are as follows:

Fetch The CPU's control unit fetches, from main memory, the next instruction in the sequence of program instructions.

Decode The instruction is encoded in the form of a number. The control unit decodes the instruction and generates an electronic signal.

Execute The signal is routed to the appropriate component of the computer (such as the ALU, a disk drive, or some other device). The signal causes the component to perform an operation.

These steps are repeated as long as there are instructions to perform.

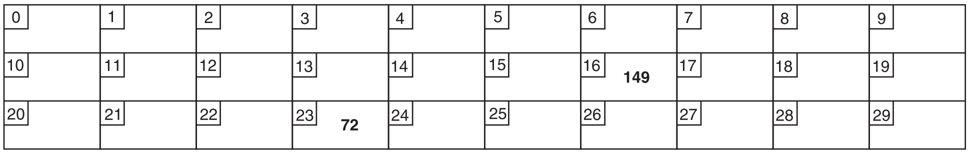
Main Memory

Commonly known as *random access memory*, or *RAM*, the computer's main memory is a device that holds information. Specifically, RAM holds the sequences of instructions in the programs that are running and the data those programs are using.

Memory is divided into sections that hold an equal amount of data. Each section is made of eight "switches" that may be either on or off. A switch in the on position usually represents the number 1, whereas a switch in the off position usually represents the number 0. The computer stores data by setting the switches in a memory location to a pattern that represents a character or a number. Each of these switches is known as a *bit*, which stands for *binary digit*. Each section of memory, which is a collection of eight bits, is known as a *byte*. Each byte is assigned a unique number known as an *address*. The addresses are ordered from lowest to highest. A byte is identified by its address in much the same way a post office box is identified by an address. Figure 1-3 shows a series of bytes with their addresses. In the illustration, sample data is stored in memory. The number 149 is stored in the byte at address 16, and the number 72 is stored in the byte at address 23.

RAM is usually a volatile type of memory, used only for temporary storage. When the computer is turned off, the contents of RAM are erased.

Figure 1-3 Memory bytes and their addresses



0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16 149	17	18	19
20	21	22	23 72	24	25	26	27	28	29

Secondary Storage

Secondary storage is a type of memory that can hold data for long periods of time—even when there is no power to the computer. Frequently used programs are stored in secondary memory and loaded into main memory as needed. Important data, such as word processing documents, payroll data, and inventory figures, is saved to secondary storage as well.

The most common type of secondary storage device is the *disk drive*. A traditional disk drive stores data by magnetically encoding it onto a spinning circular disk. *Solid state drives*, which store data in solid-state memory, are increasingly becoming popular. A solid-state drive has no moving parts, and operates faster than a traditional disk drive. Most computers have some sort of secondary storage device, either a traditional disk drive or a solid-state drive, mounted inside their case. External drives are also available, which connect to one of the computer's communication ports. External drives can be used to create backup copies of important data or to move data to another computer.

In addition to external drives, many types of devices have been created for copying data, and for moving it to other computers. *Universal Serial Bus drives,* or *USB drives* are small devices that plug into the computer's USB (Universal Serial Bus) port, and appear to the system as a disk drive. These drives do not actually contain a disk, however. They store data in a special type of memory known as *flash memory*. USB drives are inexpensive, reliable, and small enough to be carried in your pocket.

Optical devices such as the *CD* (compact disc) and the *DVD* (digital versatile disc) are also popular for data storage. Data is not recorded magnetically on an optical disc, but is encoded as a series of pits on the disc surface. CD and DVD drives use a laser to detect the pits and thus read the encoded data. Optical discs hold large amounts of data, and because recordable CD and DVD drives are now commonplace, they make a good medium for creating backup copies of data.

Input Devices

Input is any data the computer collects from the outside world. The device that collects the data and sends it to the computer is called an *input device*. Common input devices are the keyboard, mouse, scanner, and digital camera. Disk drives, optical drives, and USB drives can also be considered input devices because programs and data are retrieved from them and loaded into the computer's memory.

Output Devices

Output is any data the computer sends to the outside world. It might be a sales report, a list of names, or a graphic image. The data is sent to an output device, which formats and presents it. Common output devices are monitors and printers. Disk drives, USB drives, and CD recorders can also be considered output devices because the CPU sends data to them to be saved.

Software

As previously mentioned, software refers to the programs that run on a computer. There are two general categories of software: operating systems and application software. An operating system is a set of programs that manages the computer's hardware devices and controls their processes. Most all modern operating systems are multitasking, which means they are capable of running multiple programs at once. Through a technique called time sharing, a multitasking system divides the allocation of hardware resources and the attention of the CPU among all the executing programs. UNIX, Linux, Mac OS, and Windows are multitasking operating systems.

Application software refers to programs that make the computer useful to the user. These programs solve specific problems or perform general operations that satisfy the needs of the user. Word processing, spreadsheet, and database packages are all examples of application software.

Checkpoint

MyProgrammingLab™ *www.myprogramminglab.com*

1.1 Why is the computer used by so many different people, in so many different professions?

1.2 List the five major hardware components of a computer system.

1.3 Internally, the CPU consists of what two units?

1.4 Describe the steps in the fetch/decode/execute cycle.

1.5 What is a memory address? What is its purpose?

1.6 Explain why computers have both main memory and secondary storage.

1.7 What does the term *multitasking* mean?

1.4 Programming Languages

CONCEPT: A program is a set of instructions a computer follows in order to perform a task. A programming language is a special language used to write computer programs.

What Is a Program?

Computers are designed to follow instructions. A computer program is a set of instructions that enable the computer to solve a problem or perform a task. For example, suppose we want the computer to calculate someone's gross pay. The following is a list of things the computer should do to perform this task.

1. Display a message on the screen: "How many hours did you work?"
2. Allow the user to enter the number of hours worked.
3. Once the user enters a number, store it in memory.
4. Display a message on the screen: "How much do you get paid per hour?"
5. Allow the user to enter an hourly pay rate.
6. Once the user enters a number, store it in memory.
7. Once both the number of hours worked and the hourly pay rate are entered, multiply the two numbers and store the result in memory.
8. Display a message on the screen that shows the amount of money earned. The message must include the result of the calculation performed in Step 7.

Collectively, these instructions are called an *algorithm*. An algorithm is a set of well-defined steps for performing a task or solving a problem. Notice that these steps are sequentially ordered. Step 1 should be performed before Step 2, and so forth. It is important that these instructions be performed in their proper sequence.

Although you and I might easily understand the instructions in the pay-calculating algorithm, it is not ready to be executed on a computer. A computer's CPU can only process instructions that are written in *machine language*. If you were to look at a machine language program, you would see a stream of binary numbers (numbers consisting of only 1s and 0s). The binary numbers form machine language instructions, which the CPU interprets as commands. Here is an example of what a machine language instruction might look like:

```
1011010000000101
```

As you can imagine, the process of encoding an algorithm in machine language is very tedious and difficult. In addition, each different type of CPU has its own machine language. If you wrote a machine language program for computer A and then wanted to run it on computer B, which has a different type of CPU, you would have to rewrite the program in computer B's machine language.

Programming languages, which use words instead of numbers, were invented to ease the task of programming. A program can be written in a programming language, which is much easier to understand than machine language, and then translated into machine language. Programmers use software to perform this translation. Many programming languages have been created. Table 1-1 lists a few of the well-known ones.

Table 1-1 Programming languages

Language	Description
BASIC	Beginners All-purpose Symbolic Instruction Code is a general-purpose, procedural programming language. It was originally designed to be simple enough for beginners to learn.
FORTRAN	FORmula TRANslator is a procedural language designed for programming complex mathematical algorithms.
COBOL	Common Business-Oriented Language is a procedural language designed for business applications.
Pascal	Pascal is a structured, general-purpose, procedural language designed primarily for teaching programming.
C	C is a structured, general-purpose, procedural language developed at Bell Laboratories.
C+ +	Based on the C language, C+ + offers object-oriented features not found in C. C+ + was also invented at Bell Laboratories.
C#	Pronounced "C sharp." It is a language invented by Microsoft for developing applications based on the Microsoft .NET platform.
Java	Java is an object-oriented language invented at Sun Microsystems, and is now owned by Oracle. It may be used to develop stand-alone applications that operate on a single computer, or applications that run over the Internet from a Web server.
JavaScript	JavaScript is a programming language that can be used in a Web site to perform simple operations. Despite its name, JavaScript is not related to Java.
Perl	A general-purpose programming language used widely on Internet servers.
PHP	A programming language used primarily for developing Web server applications and dynamic Web pages.
Python	Python is an object-oriented programming language used in both business and academia. Many popular Web sites contain features developed in Python.
Ruby	Ruby is a simple but powerful object-oriented programming language. It can be used for a variety of purposes, from small utility programs to large Web applications.
Visual Basic	Visual Basic is a Microsoft programming language and software development environment that allows programmers to create Windows-based applications quickly.

A History of Java

In 1991 a team was formed at Sun Microsystems (a company that is now owned by Oracle) to speculate about the important technological trends that might emerge in the near future. The team, which was named the Green Team, concluded that computers would merge with consumer appliances. Their first project was to develop a handheld device named *7 (pronounced star seven) that could be used to control a variety of home entertainment devices. For the unit to work, it had to use a programming language that could be processed by all the devices it controlled. This presented a problem because different brands of consumer devices use different processors, each with its own machine language.

Because no such universal language existed, James Gosling, the team's lead engineer, created one. Programs written in this language, which was originally named Oak, were not translated into the machine language of a specific processor, but were translated into an intermediate language known as *byte code*. Another program would then translate the byte code into machine language that could be executed by the processor in a specific consumer device.

Unfortunately, the technology developed by the Green Team was ahead of its time. No customers could be found, mostly because the computer-controlled consumer appliance industry was just beginning. But rather than abandoning their hard work and moving on to other projects, the team saw another opportunity: the Internet. The Internet is a perfect environment for a universal programming language such as Oak. It consists of numerous different computer platforms connected together in a single network.

To demonstrate the effectiveness of its language, which was renamed Java, the team used it to develop a Web browser. The browser, named HotJava, was able to download and run small Java programs known as applets. This gave the browser the capability to display animation and interact with the user. HotJava was demonstrated at the 1995 SunWorld conference before a wowed audience. Later the announcement was made that Netscape would incorporate Java technology into its Navigator browser. Other Internet companies rapidly followed, increasing the acceptance and the influence of the Java language. Today, Java is very popular for developing not only applets for developing Web applications, mobile apps, and desktop applications.

1.5 What Is a Program Made Of?

CONCEPT: There are certain elements that are common to all programming languages.

Language Elements

All programming languages have some things in common. Table 1-2 lists the common elements you will find in almost every language.

Table 1-2 The common elements of a programming language

Language Element	Description
Key Words	These are words that have a special meaning in the programming language. They may be used for their intended purpose only. Key words are also known as reserved words.
Operators	Operators are symbols or words that perform operations on one or more operands. An operand is usually an item of data, such as a number.
Punctuation	Most programming languages require the use of punctuation characters. These characters serve specific purposes, such as marking the beginning or ending of a statement, or separating items in a list.
Programmer-Defined Names	Unlike key words, which are part of the programming language, these are words or names that are defined by the programmer. They are used to identify storage locations in memory and parts of the program that are created by the programmer. Programmer-defined names are often called identifiers.
Syntax	These are rules that must be followed when writing a program. Syntax dictates how key words and operators may be used, and where punctuation symbols must appear.

Let's look at an example Java program and identify an instance of each of these elements. Code Listing 1-1 shows the code listing with each line numbered.

NOTE: The line numbers are not part of the program. They are included to help point out specific parts of the program.

Code Listing 1-1 `Payroll.java`

```
 1  public class Payroll
 2  {
 3     public static void main(String[] args)
 4     {
 5        int hours = 40;
 6        double grossPay, payRate = 25.0;
 7
 8        grossPay = hours * payRate;
 9        System.out.println("Your gross pay is $" + grossPay);
10     }
11  }
```

Key Words (Reserved Words)

Two of Java's key words appear in line 1: `public` and `class`. In line 3, the words `public`, `static`, and `void` are all key words. The words `int` in line 5 and `double` in line 6 are also key words. These words, which are always written in lowercase, each have a special meaning in Java and can only be used for their intended purpose. As you will see, the programmer is allowed to make up his or her own names for certain things in a program. Key words, however, are reserved and cannot be used for anything other than their designated purpose. Part of learning a programming language is learning the commonly used key words, what they mean, and how to use them.

Table 1-3 shows a list of the Java key words[1].

Table 1-3 The Java key words

`abstract`	`const`	`final`	`int`	`public`	`throw`
`assert`	`continue`	`finally`	`interface`	`return`	`throws`
`boolean`	`default`	`float`	`long`	`short`	`transient`
`break`	`do`	`for`	`native`	`static`	`true`
`byte`	`double`	`goto`	`new`	`strictfp`	`try`
`case`	`else`	`if`	`null`	`super`	`void`
`catch`	`enum`	`implements`	`package`	`switch`	`volatile`
`char`	`extends`	`import`	`private`	`synchronized`	`while`
`class`	`false`	`instanceof`	`protected`	`this`	

Programmer-Defined Names

The words `hours`, `payRate`, and `grossPay` that appear in the program in lines 5, 6, 8, and 9 are programmer-defined names. They are not part of the Java language but are names made up by the programmer. In this particular program, these are the names of variables. As you will learn later in this chapter, variables are the names of memory locations that may hold data.

Operators

In line 8 the following line appears:

```
grossPay = hours * payRate;
```

The `=` and `*` symbols are both operators. They perform operations on items of data, known as operands. The `*` operator multiplies its two operands, which in this example are the variables `hours` and `payRate`. The `=` symbol is called the assignment operator. It takes the value of the expression that appears at its right and stores it in the variable whose name appears at its left. In this example, the `=` operator stores in the `grossPay` variable the result of the `hours` variable multiplied by the `payRate` variable. In other words, the statement says, "the `grossPay` variable is assigned the value of `hours` times `payRate`."

[1] Java 9 also introduces a set of *restricted words* that are treated as key words under certain circumstances. See Appendix C for the full list.

Punctuation

Notice that lines 5, 6, 8, and 9 end with a semicolon. A semicolon in Java is similar to a period in English: It marks the end of a complete sentence (or statement, as it is called in programming jargon). Semicolons do not appear at the end of every line in a Java program, however. There are rules that govern where semicolons are required and where they are not. Part of learning Java is learning where to place semicolons and other punctuation symbols.

Lines and Statements

Often, the contents of a program are thought of in terms of lines and statements. A *line* is just that—a single line as it appears in the body of a program. Code Listing 1-1 is shown with each of its lines numbered. Most of the lines contain something meaningful; however, line 7 is empty. Blank lines are only used to make a program more readable.

A statement is a complete instruction that causes the computer to perform some action. Here is the statement that appears in line 9 of Code Listing 1-1:

```
System.out.println("Your gross pay is $" + grossPay);
```

This statement causes the computer to display a message on the screen. Statements can be a combination of key words, operators, and programmer-defined names. Statements often occupy only one line in a program, but sometimes they are spread out over more than one line.

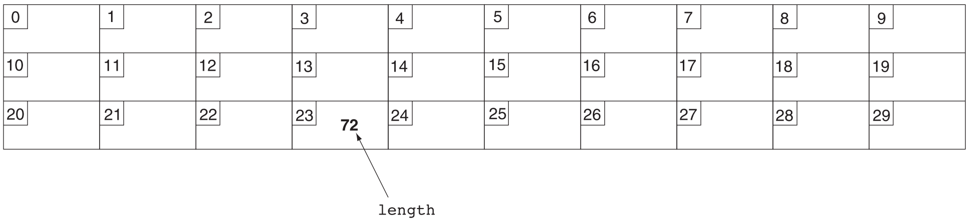
Variables

The most fundamental way that a Java program stores an item of data in memory is with a variable. A *variable* is a named storage location in the computer's memory. The data stored in a variable may change while the program is running (hence the name "variable"). Notice that in Code Listing 1-1 the programmer-defined names `hours`, `payRate`, and `grossPay` appear in several places. All three of these are the names of variables. The `hours` variable is used to store the number of hours the user has worked. The `payRate` variable stores the user's hourly pay rate. The `grossPay` variable holds the result of `hours` multiplied by `payRate`, which is the user's gross pay.

Variables are symbolic names made up by the programmer that represent locations in the computer's RAM. When data is stored in a variable, it is actually stored in RAM. Assume that a program has a variable named `length`. Figure 1-4 illustrates the way the variable name represents a memory location.

In Figure 1-4, the variable `length` is holding the value 72. The number 72 is actually stored in RAM at address 23, but the name `length` symbolically represents this storage location. If it helps, you can think of a variable as a box that holds data. In Figure 1-4, the number 72 is stored in the box named `length`. Only one item may be stored in the box at any given time. If the program stores another value in the box, it will take the place of the number 72.

Figure 1-4 A variable name represents a location in memory



The Compiler and the Java Virtual Machine

When a Java program is written, it must be typed into the computer and saved to a file. A *text editor*, which is similar to a word processing program, is used for this task. The Java programming statements written by the programmer are called *source code*, and the file they are saved in is called a *source file*. Java source files end with the *.java* extension.

After the programmer saves the source code to a file, he or she runs the Java compiler. A *compiler* is a program that translates source code into an executable form. During the translation process, the compiler uncovers any syntax errors that may be in the program. *Syntax errors* are mistakes that the programmer has made that violate the rules of the programming language. These errors must be corrected before the compiler can translate the source code. Once the program is free of syntax errors, the compiler creates another file that holds the translated instructions.

Most programming language compilers translate source code directly into files that contain machine language instructions. These are called *executable files* because they may be executed directly by the computer's CPU. The Java compiler, however, translates a Java source file into a file that contains byte code instructions. Byte code instructions are not machine language, and therefore cannot be directly executed by the CPU. Instead, they are executed by the *Java Virtual Machine* (JVM). The JVM is a program that reads Java byte code instructions and executes them as they are read. For this reason, the JVM is often called an interpreter, and Java is often referred to as an interpreted language. Figure 1-5 illustrates the process of writing a Java program, compiling it to byte code, and running it.

Although Java byte code is not machine language for a CPU, it can be considered as machine language for the JVM. You can think of the JVM as a program that simulates a computer whose machine language is Java byte code.

Portability

The term *portable* means that a program may be written on one type of computer and then run on a wide variety of computers, with little or no modification necessary. Because Java byte code is the same on all computers, compiled Java programs are highly portable. In fact, a compiled Java program may be run on any computer that has a Java Virtual Machine. Figure 1-6 illustrates the concept of a compiled Java program running on Windows, Linux, Mac, and UNIX computers.

With most other programming languages, portability is achieved by the creation of a compiler for each type of computer that the language is to run on. For example, in order for the C+ + language to be supported by Windows, Linux, and Mac computers, a separate C+ + compiler must be created for each of those environments. Compilers are very complex programs, and more difficult to develop than interpreters. For this reason, a JVM has been developed for many types of computers.

Figure 1-5 Program development process

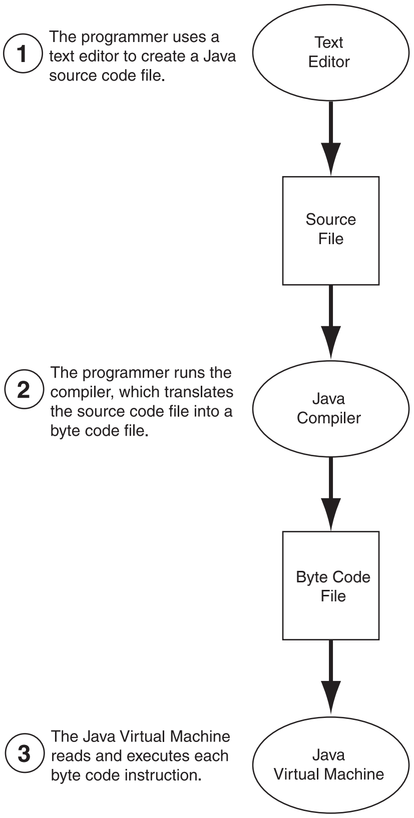
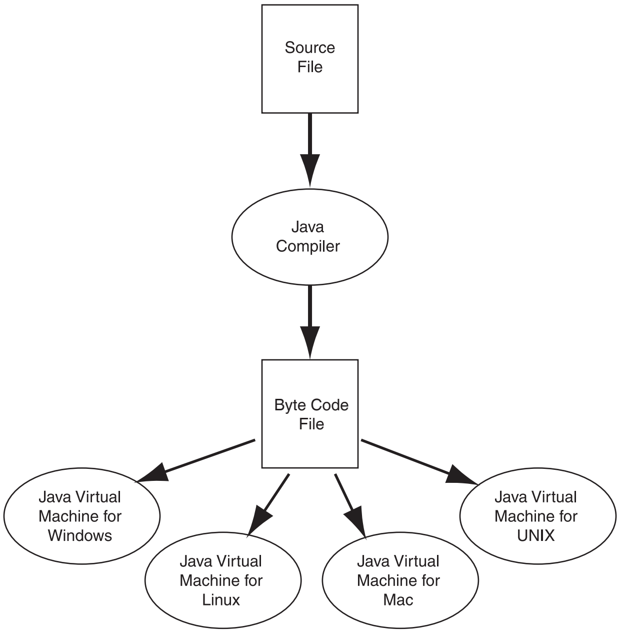

1
The programmer uses a text editor to create a Java source code file.
Text Editor
Source File
2
The programmer runs the compiler, which translates the source code file into a byte code file.
Java Compiler
Byte Code File
3
The Java Virtual Machine reads and executes each byte code instruction.
Java Virtual Machine

Figure 1-6 Java byte code may be run on any computer with a Java Virtual Machine


Source File
Java Compiler
Byte Code File
Java Virtual Machine for Windows
Java Virtual Machine for Linux
Java Virtual Machine for Mac
Java Virtual Machine for UNIX

Java Software Editions

The software that you use to create Java programs is referred to as the *JDK* (Java Development Kit) or the *SDK* (Software Development Kit). There are the following different editions of the JDK available from Oracle:

- *Java SE*—The Java Standard Edition provides all the essential software tools necessary for writing Java applications.
- *Java EE*—The Java Enterprise Edition provides tools for creating large business applications that employ servers and provide services over the Web.
- *Java ME*—The Java Micro Edition provides a small, highly optimized runtime environment for consumer products such as cell phones, pagers, and appliances.

These editions of Java may be downloaded from Oracle by going to:

http://java.oracle.com

NOTE: You can follow the instructions in Appendix D, which can be downloaded from the book's companion Web site, to install the JDK on your system. You can access the book's companion Web site by going to www.pearsonhighered.com/gaddis.

Compiling and Running a Java Program

Compiling a Java program is a simple process. Once you have installed the JDK, go to your operating system's command prompt.

TIP: In Windows click Start, go to All Programs, and then go to Accessories. Click Command Prompt on the Accessories menu. A command prompt window should open.

VideoNote
Compiling and Running a Java Program

At the operating system command prompt, make sure you are in the same directory or folder where the Java program that you want to compile is located. Then, use the `javac` command, in the following form:

```
javac  Filename
```

`Filename` is the name of a file that contains the Java source code. As mentioned earlier, this file has the *.java* extension. For example, if you want to compile the *Payroll.java* file, you would execute the following command:

```
javac Payroll.java
```

This command runs the compiler. If the file contains any syntax errors, you will see one or more error messages and the compiler will not translate the file to byte code. When this happens you must open the source file in a text editor and fix the error. Then you can run the compiler again. If the file has no syntax errors, the compiler will translate it to byte code. Byte code is stored in a file with the *.class* extension, so the byte code for the *Payroll.java* file will be stored in *Payroll.class*, which will be in the same directory or folder as the source file.

To run the Java program, you use the `java` command in the following form:

```
java  ClassFilename
```

`ClassFilename` is the name of the *.class* file that you wish to execute; however, you do not type the *.class* extension. For example, to run the program that is stored in the *Payroll.class* file, you would enter the following command:

```
java Payroll
```

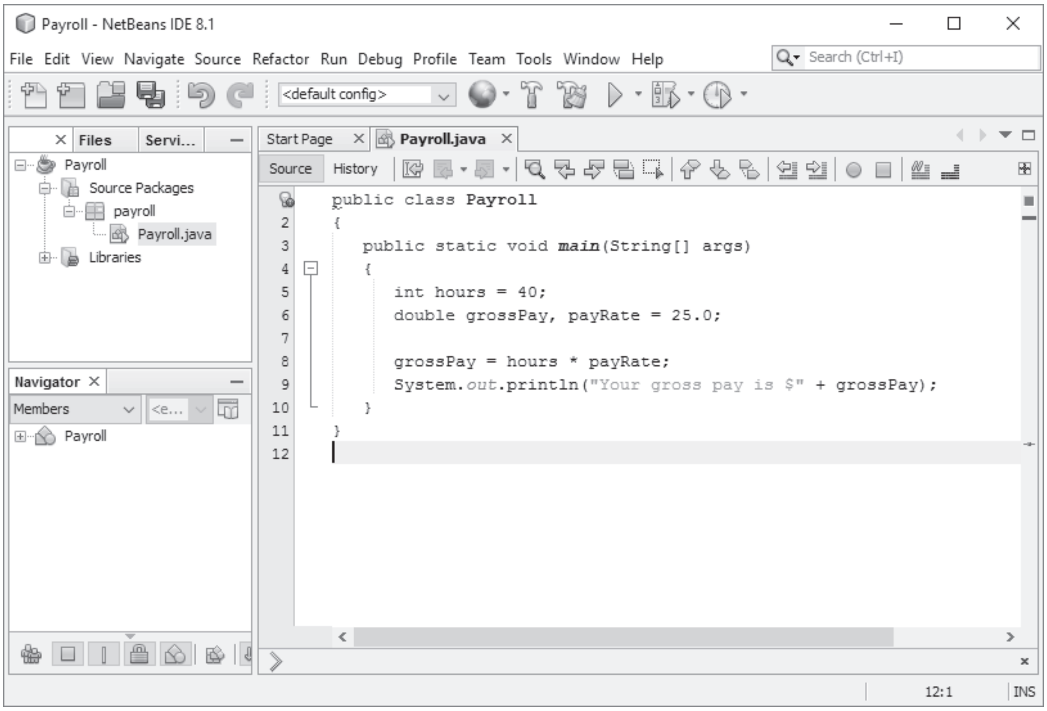
This command runs the Java interpreter (the JVM) and executes the program.

Integrated Development Environments

VideoNote
Using an IDE

In addition to the command prompt programs, there are also several Java integrated development environments (IDEs). These environments consist of a text editor, compiler, debugger, and other utilities integrated into a package with a single set of menus. A program is compiled and executed with a single click of a button, or by selecting a single item from a menu. Figure 1-7 shows a screen from the NetBeans IDE.

Figure 1-7 An integrated development environment (IDE) (Oracle Corporate Counsel)



```
public class Payroll
{
   public static void main(String[] args)
   {
      int hours = 40;
      double grossPay, payRate = 25.0;

      grossPay = hours * payRate;
      System.out.println("Your gross pay is $" + grossPay);
   }
}
```

Checkpoint

MyProgrammingLab™ *www.myprogramminglab.com*

1.8 Describe the difference between a key word and a programmer-defined symbol.

1.9 Describe the difference between operators and punctuation symbols.

1.10 Describe the difference between a program line and a statement.

1.11 Why are variables called “variable”?

1.12 What happens to a variable’s current contents when a new value is stored there?

1.13 What is a compiler?

1.14 What is a syntax error?

1.15 What is byte code?

1.16 What is the JVM?

1.6 The Programming Process

CONCEPT: The programming process consists of several steps, which include design, creation, testing, and debugging activities.

Now that you have been introduced to what a program is, it's time to consider the process of creating a program. Quite often when inexperienced students are given programming assignments, they have trouble getting started because they don't know what to do first. If you find yourself in this dilemma, the following steps may help.

1. Clearly define what the program is to do.
2. Visualize the program running on the computer.
3. Use design tools to create a model of the program.
4. Check the model for logical errors.
5. Enter the code and compile it.
6. Correct any errors found during compilation. Repeat Steps 5 and 6 as many times as necessary.
7. Run the program with test data for input.
8. Correct any runtime errors found while running the program. Repeat Steps 5 through 8 as many times as necessary.
9. Validate the results of the program.

These steps emphasize the importance of planning. Just as there are good ways and bad ways to paint a house, there are good ways and bad ways to create a program. A good program always begins with planning. With the pay-calculating algorithm that was presented earlier in this chapter serving as our example, let's look at each of the steps in more detail.

1. Clearly define what the program is to do

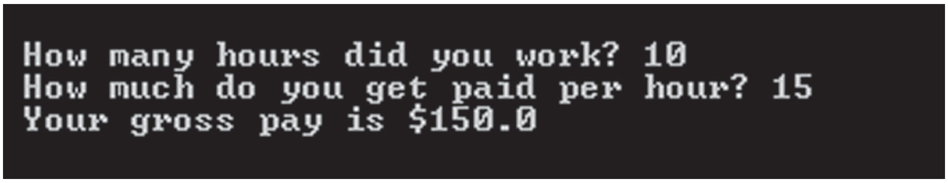
This step commonly requires you to identify the purpose of the program, the data that is to be input, the processing that is to take place, and the desired output. Let's examine each of these requirements for the pay-calculating algorithm.

Purpose	To calculate the user's gross pay.
Input	Number of hours worked, hourly pay rate.
Process	Multiply number of hours worked by hourly pay rate. The result is the user's gross pay.
Output	Display a message indicating the user's gross pay.

2. Visualize the program running on the computer

Before you create a program on the computer, you should first create it in your mind. Try to imagine what the computer screen will look like while the program is running. If it helps, draw pictures of the screen, with sample input and output, at various points in the program. For instance, Figure 1-8 shows the screen we might want produced by a program that implements the pay-calculating algorithm.

Figure 1-8 Screen produced by the pay-calculating algorithm



```
How many hours did you work? 10
How much do you get paid per hour? 15
Your gross pay is $150.0
```

In this step, you must put yourself in the shoes of the user. What messages should the program display? What questions should it ask? By addressing these concerns, you can determine most of the program's output.

3. Use design tools to create a model of the program

While planning a program, the programmer uses one or more design tools to create a model of the program. For example, *pseudocode* is a cross between human language and a programming language and is especially helpful when designing an algorithm. Although the computer can't understand pseudocode, programmers often find it helpful to write an algorithm in a language that's "almost" a programming language, but still very similar to natural language. For example, here is pseudocode that describes the pay-calculating algorithm:

```
Get payroll data.
Calculate gross pay.
Display gross pay.
```

Although this pseudocode gives a broad view of the program, it doesn't reveal all the program's details. A more detailed version of the pseudocode follows:

```
Display "How many hours did you work?"
Input hours.
Display "How much do you get paid per hour?"
Input rate.
Store the value of hours times rate in the pay variable.
Display the value in the pay variable.
```

Notice that the pseudocode uses statements that look more like commands than the English statements that describe the algorithm in Section 1.4. The pseudocode even names variables and describes mathematical operations.

4. Check the model for logical errors

Logical errors are mistakes that cause the program to produce erroneous results. Once a model of the program is assembled, it should be checked for these errors. For example, if pseudocode is used, the programmer should trace through it, checking the logic of each step. If an error is found, the model can be corrected before the next step is attempted.

5. Enter the code and compile it

Once a model of the program has been created, checked, and corrected, the programmer is ready to write source code on the computer. The programmer saves the source code to a file and begins the process of compiling it. During this step the compiler will find any syntax errors that may exist in the program.

6. Correct any errors found during compilation. Repeat Steps 5 and 6 as many times as necessary

If the compiler reports any errors, they must be corrected. Steps 5 and 6 must be repeated until the program is free of compile-time errors.

7. Run the program with test data for input

Once an executable file is generated, the program is ready to be tested for runtime errors. A runtime error is an error that occurs while the program is running. These are usually logical errors, such as mathematical mistakes.

Testing for runtime errors requires that the program be executed with sample data or sample input. The sample data should be such that the correct output can be predicted. If the program does not produce the correct output, a logical error is present in the program.

8. Correct any runtime errors found while running the program. Repeat Steps 5 through 8 as many times as necessary

When runtime errors are found in a program, they must be corrected. You must identify the step where the error occurred and determine the cause. If an error is a result of incorrect logic (such as an improperly stated math formula), you must correct the statement or statements involved in the logic. If an error is due to an incomplete understanding of the program requirements, then you must restate the program purpose and modify the program model and source code. The program must then be saved, recompiled, and retested. This means Steps 5 though 8 must be repeated until the program reliably produces satisfactory results.

9. Validate the results of the program

When you believe you have corrected all the runtime errors, enter test data and determine whether the program solves the original problem.

Software Engineering

The field of software engineering encompasses the whole process of crafting computer software. It includes designing, writing, testing, debugging, documenting, modifying, and maintaining complex software development projects. Like traditional engineers, software engineers use a number of tools in their craft. Here are a few examples:

- Program specifications
- Diagrams of screen output
- Diagrams representing the program components and the flow of data
- Pseudocode
- Examples of expected input and desired output
- Special software designed for testing programs

Most commercial software applications are large and complex. Usually a team of programmers, not a single individual, develops them. It is important that the program requirements be thoroughly analyzed and divided into subtasks that are handled by individual teams, or individuals within a team.

Checkpoint

MyProgrammingLab™ *www.myprogramminglab.com*

1.17 What four items should you identify when defining what a program is to do?

1.18 What does it mean to “visualize a program running”? What is the value of such an activity?

1.19 What is pseudocode?

1.20 Describe what a compiler does with a program’s source code.

1.21 What is a runtime error?

1.22 Is a syntax error (such as misspelling a key word) found by the compiler or when the program is running?

1.23 What is the purpose of testing a program with sample data or input?

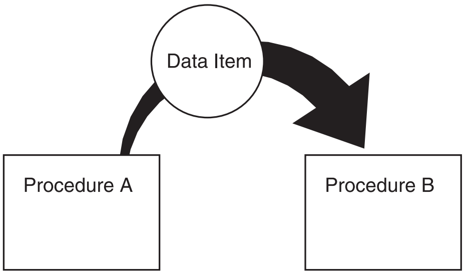
1.7 Object-Oriented Programming

CONCEPT: Java is an object-oriented programming (OOP) language. OOP is a method of software development that has its own practices, concepts, and vocabulary.

There are primarily two methods of programming in use today: procedural and object-oriented. The earliest programming languages were procedural, meaning a program was made of one or more procedures. A *procedure* is a set of programming statements that, together, perform a specific task. The statements might gather input from the user, manipulate data stored in the computer’s memory, and perform calculations or any other operation necessary to complete the procedure’s task.

Procedures typically operate on data items that are separate from the procedures. In a procedural program, the data items are commonly passed from one procedure to another, as shown in Figure 1-9.

Figure 1-9 Data is passed among procedures



As you might imagine, the focus of procedural programming is on the creation of procedures that operate on the program's data. The separation of data and the code that operates on the data often leads to problems, however. For example, the data is stored in a particular format, which consists of variables and more complex structures that are created from variables. The procedures that operate on the data must be designed with that format in mind. But, what happens if the format of the data is altered? Quite often, a program's specifications change, resulting in a redesigned data format. When the structure of the data changes, the code that operates on the data must also be changed to accept the new format. This results in added work for programmers and a greater opportunity for bugs to appear in the code.

This has helped influence the shift from procedural programming to object-oriented programming (OOP). Whereas procedural programming is centered on creating procedures, object-oriented programming is centered on creating objects. An object is a software entity that contains data and procedures. The data contained in an object is known as the object's *attributes*. The procedures, or behaviors, that an object performs are known as the object's *methods*. The object is, conceptually, a self-contained unit consisting of data (attributes) and procedures (methods). This is illustrated in Figure 1-10.

OOP addresses the problem of code/data separation through encapsulation and data hiding. *Encapsulation* refers to the combining of data and code into a single object. *Data hiding* refers to an object's ability to hide its data from code that is outside the object. Only the object's methods may then directly access and make changes to the object's data. An object typically hides its data, but allows outside code to access the methods that operate on the data. As shown in Figure 1-11, the object's methods provide programming statements outside the object with indirect access to the object's data.

When an object's internal data is hidden from outside code and access to that data is restricted to the object's methods, the data is protected from accidental corruption. In addition, the programming code outside the object does not need to know about the format or internal structure of the object's data. The code only needs to interact with the object's methods.

Figure 1-10 An object contains data and procedures

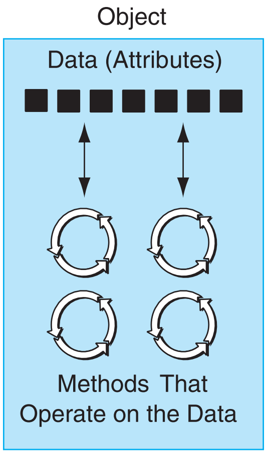
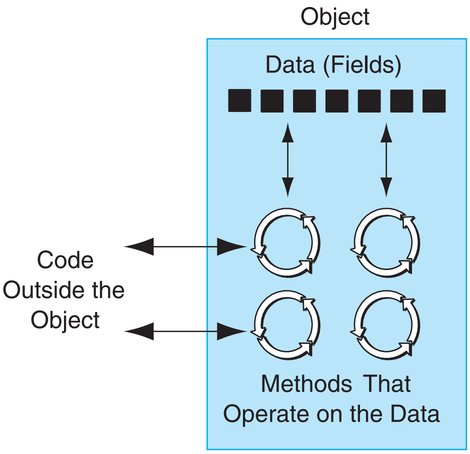


Figure 1-11 Code outside the object interacts with the object's methods



When a programmer changes the structure of an object's internal data, he or she also modifies the object's methods so they may properly operate on the data. The way in which outside code interacts with the methods, however, does not change.

These are just a few of the benefits of object-oriented programming. Because Java is fully object-oriented, you will learn much more about OOP practices, concepts, and terms as you progress through this book.

Checkpoint

MyProgrammingLab™ *www.myprogramminglab.com*

1.24 In procedural programming, what two parts of a program are typically separated?

1.25 What are an object's attributes?

1.26 What are an object's methods?

1.27 What is encapsulation?

1.28 What is data hiding?

Review Questions and Exercises

Multiple Choice

1. This part of the computer fetches instructions, carries out the operations commanded by the instructions, and produces some outcome or resultant information.
   a. memory
   b. CPU
   c. secondary storage
   d. input device
2. A byte is made up of eight
   a. CPUs
   b. addresses
   c. variables
   d. bits
3. Each byte is assigned a unique
   a. address
   b. CPU
   c. bit
   d. variable
4. This type of memory can hold data for long periods of time—even when there is no power to the computer.
   a. RAM
   b. primary storage
   c. secondary storage
   d. CPU storage

5. If you were to look at a machine language program, you would see _______.
   a. Java source code
   b. a stream of binary numbers
   c. English words
   d. circuits
6. These are words that have a special meaning in the programming language.
   a. punctuation
   b. programmer-defined names
   c. key words
   d. operators
7. These are symbols or words that perform operations on one or more operands.
   a. punctuation
   b. programmer-defined names
   c. key words
   d. operators
8. These characters serve specific purposes, such as marking the beginning or ending of a statement, or separating items in a list.
   a. punctuation
   b. programmer-defined names
   c. key words
   d. operators
9. These are words or names that are used to identify storage locations in memory and parts of the program that are created by the programmer.
   a. punctuation
   b. programmer-defined names
   c. key words
   d. operators
10. These are the rules that must be followed when writing a program.
    a. syntax
    b. punctuation
    c. key words
    d. operators
11. This is a named storage location in the computer's memory.
    a. class
    b. key word
    c. variable
    d. operator
12. The Java compiler generates _______.
    a. machine code
    b. byte code
    c. source code
    d. HTML
13. JVM stands for _______.
    a. Java Variable Machine
    b. Java Variable Method
    c. Java Virtual Method
    d. Java Virtual Machine

Find the Error

1. The following pseudocode algorithm has an error. The program is supposed to ask the user for the length and width of a rectangular room, and then display the room's area. The program must multiply the width by the length to determine the area. Find the error.

```
area = width × length
Display "What is the room's width?"
Input width
Display "What is the room's length?"
Input length
Display area
```

Algorithm Workbench

Write pseudocode algorithms for the programs described as follows:

1. **Available Credit**

   A program that calculates a customer's available credit should ask the user for the following:

   - The customer's maximum amount of credit
   - The amount of credit used by the customer

   Once these items have been entered, the program should calculate and display the customer's available credit. You can calculate available credit by subtracting the amount of credit used from the maximum amount of credit.

2. **Sales Tax**

   A program that calculates the total of a retail sale should ask the user for the following:

   - The retail price of the item being purchased
   - The sales tax rate

   Once these items have been entered, the program should calculate and display the following:

   - The sales tax for the purchase
   - The total of the sale

3. **Account Balance**

   A program that calculates the current balance in a savings account must ask the user for the following:

   The starting balance
   The total dollar amount of deposits made
   The total dollar amount of withdrawals made
   The monthly interest rate

   Once the program calculates the current balance, it should be displayed on the screen.