Preface

Purpose of This Book and Its Audience

Java Illuminated, Third Edition, covers all of the material required for the successful completion of an introductory course in Java. While the focus is on the material required for the Computer Science I (CS1) and Computer Science II (CS2) curricula, students enrolled in Information Systems, Information Technology, or self-directed study courses will find the book useful as well. It has been written to provide introductory computer science students with a comprehensive overview of the fundamentals of programming using Java as the teaching language. In addition, the book presents other topics of interest, including graphical user interfaces (GUI), data structures, file input and output, and applets.

Throughout the book, we take an "active learning" approach to presenting the material. Instead of merely presenting the concepts to students in a one-sided, rote manner, we ask them to take an active role in their understanding of the language through the use of numerous interactive examples, exercises, and projects.

Coverage and Approach

Our approach is to teach object-oriented programming in a progressive manner. We start in Chapter 1 by presenting an overview of object-oriented programming. In Chapter 3, we delve a little deeper into the concepts of classes and objects and introduce the student to many of the useful classes in the Java class library. Our emphasis at this point is on using

classes; we teach the student how to read APIs in order to determine how to instantiate objects and call methods of the classes. In Chapter 7, we move on to designing user-defined classes, and in Chapter 10, we present inheritance, polymorphism, and interfaces. Throughout the book, we present concepts in an object-oriented context.

Our philosophy is to emphasize good software engineering practices by focusing on designing and writing correct, maintainable programs. As such, we discuss pseudocode, testing techniques, design trade-offs, and other software engineering tips.

We teach the student basic programming techniques, such as accumulation, counting, calculating an average, finding maximum and minimum values, using flag and toggle variables, and basic searching and sorting algorithms. In doing so, we emphasize the patterns inherent in programming. Concepts are taught first, followed by fully implemented examples with source code. We promote Java standards, conventions, and methodologies.

This book supports the important features of the latest versions of Java (5, 6, and 7). The *Scanner* class is used to simplify user input from the keyboard and in reading from files. In Chapter 5, we demonstrate a new Java 7 feature: performing a *switch* statement with a *String* expression. The *enum* functionality is presented as a user-defined data type in Chapter 7. Autoboxing and unboxing concepts are introduced in Chapter 3 with the Java wrapper classes. We demonstrate generic types and the enhanced *for* loop in the Chapter 9 coverage of *ArrayLists*, and we explain how to write a class using generic types in Chapter 14.

Learning Features

Recognizing today's students' growing interest in animation and visualization, we distribute techniques for producing graphical output and animation throughout the book, starting in Chapter 4 with applets. An example using either animation or graphical output is included in most chapters. Instructors who are not interested in incorporating graphics into their curriculum can simply skip these sections. In addition, some of our examples are small games, which we find motivational for students.

In each chapter, we include one or two Programming Activities, which are designed to provide visual feedback to the students so that they can assess the correctness of their code. In most Programming Activities, we provide

a framework, usually with a graphical user interface, to which the student adds code to complete the application. The student should be able to finish the Programming Activity in about 15 to 20 minutes; thus, these activities can be used in the classroom to reinforce the topics just presented. Each Programming Activity also includes several discussion questions that test the student's understanding of the concepts the activity illustrates. The Programming Activities are also appropriate for a closed or open laboratory environment. In short, this book can be used in a traditional lecture environment, a computer-equipped classroom, or a lab environment.

In addition, we supplement each chapter with a browser-based module that animates sample code, visually illustrating the assignment of variable values, evaluation of conditions, and flow of control.

We also provide the instructor and students with an extensive variety of end-of-chapter material: multiple-choice questions, examples that ask the student to predict the output of prewritten code or to fill in missing code, debugging activities, short exercises, programming projects, technical writing assignments, and a higher-difficulty group project.

Chapter-by-Chapter Overview

The chapters are logically organized from simple to more difficult topics, while incorporating object orientation as needed, taking into account the specifics of the Java language. Here is a brief summary of the topics covered in each chapter:

Chapter 1: Introduction to Programming and the Java Language

We introduce the student to the concept of programming, first covering computer hardware and operating systems, and following with a brief evolution of programming languages, including an introduction to object-oriented programming. We explain programming basics and pseudocode as a program design technique. The student writes, compiles, and debugs their first program using an integrated development environment.

Chapter 2: Programming Building Blocks—Java Basics

In this chapter, we concentrate on working with variables and constants of primitive data types and composing arithmetic expressions. We illustrate the differences between integer and floating-point calculations and introduce operator precedence.

Chapter 3: Object-Oriented Programming, Part 1: Using Classes

Chapter 3 introduces classes from the user, or client, standpoint and discusses the benefits of encapsulation and code reuse. The student learns how to instantiate objects and call methods. We also demonstrate useful Java classes for console input and output, dialog boxes, formatting output, performing mathematical calculations, and generating random numbers.

Chapter 4: Introduction to Applets and Graphics

Chapter 4 presents several methods of the *Graphics* class that can be used to create graphical output by drawing shapes and text. The windowing graphics coordinate system is explained and using color is also explored. We demonstrate these graphics methods in applets because an applet window provides an easy-to-use palette for drawing. Instructors wishing to postpone or skip graphics coverage altogether can use as little or as much of this chapter as they desire.

Chapter 5: Flow of Control, Part 1: Selection

Various forms of the *if*, *if/else*, and *if/else if* statements are presented, along with the appropriate situations in which to use each form. We also demonstrate nested *if/else* statements and testing techniques. We begin our coverage of scope by introducing block scope. Later chapters build upon this foundation. As part of our object-oriented programming coverage, we teach the importance of comparing objects using the *equals* method. This chapter also covers the conditional operator and the *switch* statement.

Chapter 6: Flow of Control, Part 2: Looping

This is probably the most important chapter in the book. We have found that looping and repetition are the most difficult basic programming concepts for the average student to grasp. We try to ease the student's understanding of looping techniques by presenting patterns to follow in coding basic algorithms: accumulation, counting, calculating an average, and finding minimum and maximum values. We present a motivational and engaging example of repetition in the animation of a ball rolling across the screen. Looping is further explored as a tool for validation of input values. We continue our coverage of scope by illustrating the scope of variables declared within the *while* loop body and *for* loop header. We concentrate on using the *while* loop for event-controlled and sentinel-controlled repetition and the *for* loop for count-controlled looping. A large section focuses on constructing loop conditions, which is often a challenging task for the student. Sections are also provided on testing techniques for *while* loops and for *for* loops. In this chapter, we also introduce reading data from a text file using the *Scanner* class.

Chapter 7: Object-Oriented Programming, Part 2: User-Defined Classes

In this chapter, we teach the student to write classes, as well as client applications, that use the instantiated objects and call methods of the class. We present class design techniques and standard patterns for writing constructors, mutators and accessors, and the *toString, equals,* and other user-defined methods. We further explain scope in the context of class members and method parameters. We also explain how and when to use the keywords *this* and *static. Enum* is also covered as a user-defined class type. Finally, we teach the student how to use Javadoc and how to create a package.

Chapter 8: Single-Dimensional Arrays

This chapter begins with the declaration, instantiation, and initialization of single-dimensional arrays. From there, the student learns to perform the basic programming techniques (accumulation, counting, calculating an average, and finding maximum and minimum values) on array elements. We also cover arrays as instance variables of a class, and demonstrate maintaining encapsulation while accepting arrays as method parameters and returning arrays from methods. Basic searching and sorting algorithms are also presented, including sequential and binary searches and Selection and Insertion sorts.

Chapter 9: Multidimensional Arrays and the *ArrayList* Class

We focus in this chapter on two-dimensional array processing, including techniques for processing all the elements in the entire array, or the elements in a specific column or row. We also demonstrate the extra processing needed to handle arrays with rows of different lengths. A bar chart of the data in each row of the array is also demonstrated. In addition, we extrapolate the concepts from two-dimensional arrays to discuss multidimensional arrays.

We present the *ArrayList* class as an expandable array and demonstrate using classes with generic types, the enhanced *for* loop, and autoboxing and unboxing.

Chapter 10: Object-Oriented Programming, Part 3: Inheritance, Polymorphism, and Interfaces

Continuing our object-oriented programming coverage, we discuss the important concepts and benefits of inheritance and the design of class hierarchies, including abstract classes. We cover inherited members of a class, constructing objects of a subclass, adding specialization to a subclass, overriding inherited methods, and calling methods of the superclass. We discuss the trade-offs of declaring members as *protected* versus *private.* We

demonstrate polymorphism with a graphical example, and introduce the student to interfaces, which are used extensively in Graphical User Interfaces. (See Chapter 12.)

Chapter 11: Exceptions and Input/Output Operations

Recognizing that building robust applications requires error handling, we present exception handling as a tool for validating user input and recovering from errors at run time. We demonstrate handling predefined exceptions and writing user-defined exceptions.

With this knowledge, the student is ready to perform file input and output operations. We demonstrate reading and writing *Strings* and primitive data types to text files, and reading and writing objects directly to files. The *StringTokenizer* class is used to read and parse input from text files.

Chapter 12: Graphical User Interfaces

This chapter introduces the student to event-driven programming and writing event handlers for text fields, buttons, radio buttons, checkboxes, lists, combo boxes, and mouse activities. We also demonstrate panels and several layout managers for organizing GUI components, as well as how to nest components. In our examples, we illustrate how to separate the graphical user interface code from the underlying data and program logic.

Chapter 13: Recursion

Recursion is presented as a design technique, reducing the size of a problem until an easy-to-solve problem is reached. We demonstrate recursive methods with one base case and with multiple base cases, and with and without return values. Specific examples provided include computing the factorial of a number, finding the greatest common divisor, performing a binary search, determining if a phrase is a palindrome, calculating combinations, solving the Towers of Hanoi problem, and performing animation. The benefits and trade-offs of recursion versus iteration are also discussed.

Chapter 14: An Introduction to Data Structures

In this chapter, we cover data structures by exploring the concepts and implementations of various types of linked lists, stacks, and queues. We demonstrate many types and uses of linked lists: a singly linked list, a

linked list as a stack, a linked list as a queue, a doubly linked list, a sorted linked list, and a recursively defined linked list. Arrays as stacks and circular arrays as queues are also covered in detail.

We begin with a list of primitive types (*int*) and progress to a list consisting of objects of a user-defined *Player* class. Then we cover defining a class using generic types to demonstrate how a list can be defined to hold generic objects.

Chapter 15: Running Time Analysis

We explain how to evaluate the performance of an algorithm in this chapter. We explain the Big-Oh notation and orders of magnitude. Students learn various methods for deriving performance estimates: counting statements in loops, iterative, handwaving, and proof by induction analyses for recursive methods. We demonstrate how the coding of an algorithm influences its running time. Worst-case, best-case, and average-case performances are explained and illustrated.

What's New in *Java Illuminated*

In this edition, we have refined the existing material and incorporated new material as a result of feedback from instructors who have adopted our book.

We revised Chapter 2 to better introduce the concept of programming. We provide a simple, but complete program near the beginning of the chapter so that the student can get a sense of what a simple program looks like and a feel for how a program operates. We have revised both Programming Activities to be more real-world examples: converting inches to centimeters and converting temperatures between Fahrenheit and Celsius. We also introduce arithmetic operators earlier in the chapter so that the student can more quickly write a program that performs calculations. We also help the student to think through the tasks involved with designing a program, such as identifying the data, calculations, and output.

In Chapter 3, we slightly altered Example 3.20 to perform calculations on the values retrieved from the dialog boxes to demonstrate the need for converting the *String* return value to a numeric type.

In Chapter 5, we modified Example 5.14, A Simple Calculator, to use a *String* as the *switch* expression, a new feature supported in Java 7. We also slightly altered Example 5.12, Comparing Strings, so that we also test for inequality using the *equals* method.

The largest changes were made to Chapter 11. We removed some redundant classes for writing to files. Because *StringTokenizer* class is no longer recommended, we now use *Scanner* to read and parse input lines from a file.

Finally, in Chapter 14, we removed the encapsulation of list elements to be more consistent with the Collections classes in the Java Class Library.

Pedagogy

Concepts are always taught first, followed by complete, executable examples illustrating these concepts. Most examples demonstrate real-life applications so that the student can understand the need for the concept at hand. The example code is colored to better illustrate the syntax of the code and to reflect the use of colors in today's IDE tools, as shown in this example from Chapter 3:

```
1 /*  A demonstration of reading from the console using Scanner
2     Anderson, Franceschi
3 */
4
5 import java.util.Scanner;
6
7 public class DataInput
8 {
9   public static void main( String [ ] args )
10  {
11     Scanner scan = new Scanner( System.in );
12
13     System.out.print( "Enter your first name > " );
14     String firstName = scan.next( );
15     System.out.println( "Your name is " + firstName );
16
17     System.out.print( "\nEnter your age as an integer > " );
18     int age = scan.nextInt( );
19     System.out.println( "Your age is " + age );
20
21     System.out.print( "\nEnter your GPA > " );
22     float gpa = scan.nextFloat( );
23     System.out.println( "Your GPA is " + gpa );
24  }
25 }
```

EXAMPLE 3.9 Reading from the Console using *Scanner*

Figures and tables are used to illustrate or summarize the concept at hand, such as these from Chapters 6 and 7:

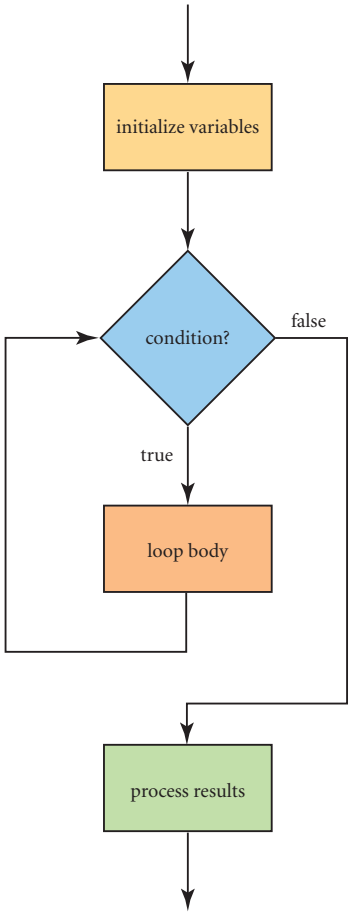

Figure 6.1
Flow of Control of a *while* Loop

TABLE 7.1 Access Modifiers

Access Modifier	Class or member can be referenced by . . .
public	methods of the same class, as well as methods of other classes
private	methods of the same class only
protected	methods in the same class, as well as methods of subclasses and methods in classes in the same package
no modifier (package access)	methods in the same package only

SOFTWARE ENGINEERING TIP

Define instance variables of a class as *private* so that only methods of the class will be able to set or change their values.

COMMON ERROR TRAP

Be sure that both operands of the logical AND and logical OR operators are *boolean* expressions. Expressions such as this: `x < y && z`, with *x*, *y*, and *z* being numeric types, are illegal. Instead, use the expression: `x < y && x < z`

In each chapter, we emphasize good design concepts using "Software Engineering Tips," such as the one to the left from Chapter 7.

We also provide "Common Error Traps," such as the one to the left from Chapter 5, to alert students against common syntax and logic errors.

In each chapter, "active learning" programming activities reinforce concepts with enjoyable, hands-on projects that provide visual feedback to the students. These activities can be done in lab-type classrooms or can be assigned as projects. A header for a Programming Activity looks like this:

6.9 Programming Activity 1: Using *while* Loops

In this activity, you will work with a sentinel-controlled *while* loop, performing this activity:

> Write a *while* loop to process the contents of a grocery cart and calculate the total price of the items. It is important to understand that, in this example, we do not know how many items are in the cart.

Supplementing each chapter, we provide a browser-based module implemented as a Flash animation on the CD-ROM, which illustrates the execution of code that implements the concepts taught in the chapter. Each movie animates a brief code sample, one line at a time, and is controlled by the user via a "Next Step" button. These modules can be beneficial for students who learn best with visual aids, graphs, illustrations, and at their own pace outside the classroom. The modules are announced in each chapter using a special icon as in the sample below.

CODE IN ACTION

To see two step-by-step illustrations of do/while loops, look for the Chapter 6 Flash movie on the CD-ROM included with this book. Click on the link for Chapter 6 to start the movie.

Graphics Coverage

Graphics are distributed throughout the book and are used to engage the student and reinforce the chapter concepts. The Graphics coordinate system, methods for drawing shapes and text, and color concepts are presented with simple applets in Chapter 4. Animation using loops is demonstrated in Chapter 6, while drawing a bull's-eye target illustrates both looping and using a toggle variable. Classes for displayable objects are presented in Chapter 7; drawing a bar chart of array data is illustrated in Chapters 8 and 9; polymorphism is demonstrated using a Tortoise and Hare Race in Chapter 10; GUIs are covered in Chapter 12; and animation using recursion is demonstrated in Chapter 13. The two figures that follow illustrate graphical examples from Chapters 7 and 8.

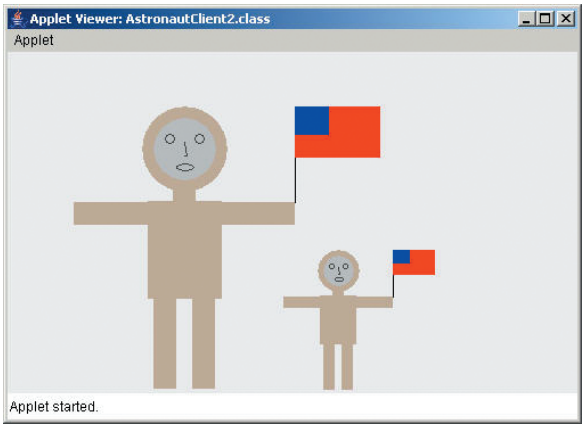

Figure 7.10
The *AstronautClient2* Window

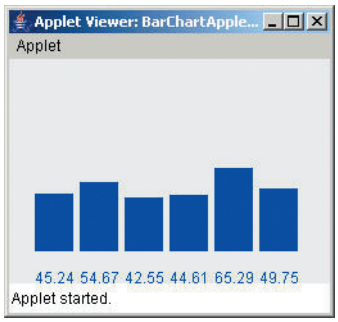

Figure 8.15
The *cellBills* Array as a Bar Chart

End-of-Chapter Exercises and Problems

A large collection of exercises and problems is proposed at the end of each chapter. Short exercises cover programming from a variety of angles: multiple choice concept questions, reading and understanding code segments, filling in some code, correcting errors, and interpreting compiler error messages to diagnose application bugs. Many programming projects are proposed with an emphasis on writing *classes*, not just a *program*. A more challenging group project is proposed in each chapter, allowing students to work as a group and develop communication skills, in accordance with recommendations from accreditation organizations. Small, essay-type questions are also proposed to enable students to acquire proficiency in technical writing and communication.

CD-ROM Accompanying This Book

Included in the CD-ROM accompanying this book are:

- Programming Activity framework code
- Full example code from each chapter
- Browser-based modules with visual step-by-step demonstrations of code execution
- Links to various Integrated Development Environments
- Link to the most recent version of Java™ 2 Standard Edition JDK

Appendices

The appendices include the following:

- Java reserved words and keywords
- Operator precedence
- Unicode character set
- Representing negative numbers
- Representing floating-point numbers
- Java classes and APIs presented in this book
- Answers to selected exercises

Instructor Resources

These materials are available to instructors on the Jones & Bartlett Learning website (http://www.jblearning.com/catalog/9781449604387/), and include

- Programming activity solution code (for instructors only)
- Answers to many end-of-chapter exercises
- PowerPoint® slides for each chapter
- Test items

Contacting the Authors

We have checked and rechecked the many technical details in this book. Despite our best efforts, however, we realize that some errors may have been missed. If you discover a technical error in the book, please contact us at JulieAustinAnderson@gmail.com or hfranceschi@capitol-college.edu. We will post any corrections on the book's website: http://www.jblearning.com/catalog/9781449604387/.