Preface

Welcome to Java™ Software Solutions for AP* Computer Science. This Advanced Placement* book matches both the AP* Computer Science topic outline and Java subset set forth by the College Board.

What's New in This Edition

We have been pleased to hear that this book has met and exceeded the needs of many instructors of AP* Computer Science courses. We appreciate your feedback and suggestions.

This third edition of Java Software Solutions for AP* Computer Science has been updated to ensure that the most recent requirements for the Advanced Placement test have been thoroughly covered. In addition, the flow and discussion of certain topics have been improved and updated.

The specific changes include:

- The coverage of primitive types in Chapter 2 now provides more detail of internal representation and a discussion of imprecision and round-off error.
- An appendix has been added covering number systems and examples of conversions between bases.
- A discussion of the Integer.MAX_VALUE and Integer.MIN_VALUE constants is now part of the Chapter 2 coverage of wrapper classes.
- The Math.random method is discussed in more detail in Chapter 2 as an alternative to using the Random class.
- Chapter 3 now includes a discussion of loop analysis.
- A new section in Chapter 5 covers Testing and Debugging.
- End-of-chapter exercises have been updated to reflect the modified and new material.

About This AP* Book

This book has been specifically designed to meet the needs of today’s AP* students and AP* teachers. Let’s hit the highlights. Specifically:

- Online sections for most chapters tie the official AP* case study in with the concepts covered in that chapter. This material is available on the book’s Web site.

*AP and Advanced Placement Program are registered trademarks of The College Board, which was not involved in the production of, and does not endorse, this product.

We utilize hundreds of example programs, fully implemented for experimentation by AP* students and AP* teachers.

Graphics are covered in an optional section (called Graphics Track) at the end of each chapter. Building on the excitement of the Web, the Graphics Track is intended to further inspire and engage AP* students.

The Self-Review Questions, Exercises, and Programming Projects have been specifically designed to adhere to various learning styles, including multiple choice, true/false, and short-answer questions.

The reference material in the appendices includes: Glossary, Unicode Character Set, Java Operators, and Number Systems.

A robust ancillary package accompanies this book and is outlined further in this preface.

The full-color design aids learning by making it easier to distinguish between various elements in code and diagrams. Full-color screen shots make the discussions of graphical interfaces more insightful and realistic.

**Cornerstones of the Text**

This text is based on the following basic ideas that we believe make for a sound introductory text:

- **True object-orientation.** A text that really teaches a solid object-oriented approach must use what we call object-speak. That is, all processing should be discussed in object-oriented terms. That does not mean, however, that the first program an AP* student sees must discuss the writing of multiple classes and methods. An AP* student should learn to use objects before learning to write them. This text uses a natural progression that culminates in the ability to design real object-oriented solutions.

- **Sound programming practices.** AP* students should not be taught how to program; they should be taught how to write good software. There’s a difference. Writing software is not a set of cookbook actions, and a good program is more than a collection of statements. This text integrates practices that serve as the foundation of good programming skills. These practices are used in all examples and are reinforced in the discussions. AP* students learn how to solve problems as well as how to implement solutions. We introduce and integrate basic software engineering techniques throughout the text.

- **Examples.** AP* students learn by example. This text is filled with fully implemented examples that demonstrate specific concepts. We have
intertwined small, readily understandable examples with larger, more realistic ones. There is a balance between graphics and nongraphical programs and between applets and applications.

- **Graphics and GUIs.** Graphics can be a great motivator for AP* students, and their use can serve as excellent examples of object-orientation. As such, we use them throughout the text in a well defined set of sections that we call the Graphics Track. This coverage includes the use of event processing and graphical user interfaces (GUIs). AP* students learn to build GUIs in the appropriate way by using a natural progression of topics. The Graphics Track can be avoided entirely for those who do not choose to use graphics.

**Chapter Breakdown**

Chapter 1 (Computer Systems) introduces computer systems in general, including basic architecture and hardware, networking, programming, and language translation. Java is introduced in this chapter, and the basics of program development are discussed. This chapter contains broad introductory material that can be covered while AP* students become familiar with their development environment.

Chapter 2 (Objects and Primitive Data) establishes the concept of objects and how they can be used. Many predefined classes from the Java standard library are explored and used. Enumerated types, primitive types, operators, and expressions are also explored.

Chapter 3 (Program Statements) covers most of the fundamental statements including conditionals, loops, and iterators. Some additional operators are introduced at this point as well. Establishing key statements at this point allows the classes of the next chapter to be fully functional and realistic.

Chapter 4 (Writing Classes) explores issues related to writing classes and methods. Topics include instance data, visibility, scope, method parameters, and return types. Method overloading is covered as well. Some of the more involved topics are deferred to or revisited in Chapter 5. The key to Chapter 4 is the many fully implemented, realistic classes that are presented as examples of class design.

Chapter 5 (Enhancing Classes) covers additional issues related to class design and revisits topics that need further exploration. Object references are revisited and carefully explored, and their impact on parameter passing is discussed. Exceptions, interfaces, and their effect on design are also covered. Finally, guidelines on designing classes, as well as testing, are given.

Chapter 6 (Arrays) contains extensive coverage of arrays and array processing. Topics include multidimensional arrays, searching, and sorting. The `ArrayList` class is explored as well, and generic types are introduced here.
Chapter 7 (Inheritance) covers class derivations and associated concepts such as class hierarchies, overriding, and polymorphism. Emphasis is put on the proper use of inheritance and its role in software design.

Chapter 8 (Recursion) covers the concept, implementation, and proper use of recursion. Several examples from various domains are used to demonstrate how recursive techniques make certain types of processing elegant. Recursive sorting algorithms are also covered.

Chapter 9 (Linear Data Structures) introduces the idea of a collection and its underlying data structure. Abstraction is revisited in this context and the linked list, queue, and stack data structures are explored.

Chapter 10 (Non-linear Data Structures) further explores dynamic data structures, including trees and heaps. Sets and maps are also introduced and hashtables are revisited.

Supplements
AP* students are welcome to visit www.pearsonhighered.com/cssupport (author: Lewis/Loftus/Cocking) for the following resources:

- Source Code to all program examples in the text.
- Case Study section for most chapters linking the concepts covered in that chapter with the official AP* case study.

The following supplements are available online for qualified AP* teachers only, at www.pearsonhighered.com/irc. Teachers may obtain a code to access the supplements below from their Pearson sales representative. If you need help finding this material, contact your Pearson sales representative. You can find your rep in the “Find your sales rep” section at www.pearsonschool.com.

- Lesson Plans with Pacing Guide.
- Solutions to all end-of-chapter exercises and programming projects.
- PowerPoint slides.
- Test Bank with powerful test generator software, includes a wealth of free response, multiple choice, and true/false questions.
- Source Code to all program examples in the text.

AP* Correlation Guide
On the next page, we present the AP* Computer Science correlation guide as given by the College Board. The column on the right shows the sections of this book where each topic is covered.
## I. OBJECT-ORIENTED PROGRAM DESIGN

The overall goal for designing a piece of software (a computer program) is to correctly solve the given problem. At the same time, this goal should encompass specifying and designing a program that is understandable, can be adapted to changing circumstances, and has the potential to be reused in whole or in part. The design process needs to be based on a thorough understanding of the problem to be solved.

### A. Program design

1. Read and understand a problem description, purpose, and goals. Sections 1.3, 3.0, 3.7
2. Apply data abstraction and encapsulation. Sections 2.1, 4.1, 7.7
3. Read and understand class specifications and relationships among the classes ("is-a," "has-a" relationships). Sections 4.5, 7.0
4. Understand and implement a given class hierarchy. Sections 7.0, 7.2
5. Identify reusable components from existing code using classes and class libraries. Sections 2.0, 7.4

### B. Class design

1. Design and implement a class. Sections 4.0–4.4, 5.0, 5.1, 5.4
2. Choose appropriate data representation and algorithms. Sections 3.7, 5.4
3. Apply functional decomposition. Sections 4.2, 4.4
4. Extend a given class using inheritance. Sections 7.0, 7.1, 7.3, 7.4

## II. PROGRAM IMPLEMENTATION

The overall goals for program implementation parallel those of program design. Classes that fill common needs should be built so that they can be reused easily in other programs. Object-oriented design is an important part of program implementation.

### A. Implementing techniques

1. Methodology
   a. Object-oriented development Section 2.0
   b. Top-down development Section 3.0
   c. Encapsulation and information hiding Section 4.1
   d. Procedural abstraction Section 4.2

### B. Programming constructs

1. Primitive types vs. objects Sections 2.0, 2.1, 2.4
2. Declaration
   a. Constant declarations Section 2.3
   b. Variable declarations Section 2.3
   c. Class declarations Section 4.1
   d. Interface declarations Section 5.3
   e. Method declarations Section 4.2
   f. Parameter declarations Section 4.2
3. Console output (System.out.print/println) Section 2.1
4. Control
   a. Methods Section 4.2
   b. Sequential Section 3.1
   c. Conditional Section 3.2
   d. Iteration Sections 3.5, 3.6
   e. Understand and evaluate recursive methods Sections 8.0–8.2
## III. PROGRAM ANALYSIS

The analysis of programs includes examining and testing programs to determine whether they correctly meet their specifications. It also includes the analysis of programs or algorithms in order to understand their time and space requirements when applied to different data sets.

### A. Testing
1. Test classes and libraries in isolation
2. Identify boundary cases and generate appropriate test data
3. Perform integration testing

### B. Debugging
1. Categorize errors: compile-time, run-time, logic
2. Identify and correct errors
3. Techniques: use a debugger, add extra output statements, hand-trace code

### C. Understand and modify existing code
Learned throughout the text

### D. Extend existing code using inheritance
Sections 7.0, 7.1, 7.3, 7.4

### E. Understand error handling
1. Understand runtime exceptions

### F. Reason about programs
1. Pre- and post-conditions
2. Assertions

### G. Analysis of algorithms
1. Informal comparisons of running times
2. Exact calculation of statement execution counts

### H. Numerical representations and limits
1. Representations of numbers in different bases
2. Limitations of finite representations (e.g., integer bounds, imprecision of floating-point representations, and round-off error)

## IV. STANDARD DATA STRUCTURES

Data structures are used to represent information within a program. Abstraction is an important theme in the development and application of data structures.

### A. Simple data types (int, boolean, double)
Section 2.4

### B. Classes
Sections 2.0, 4.1
### AP* Computer Science Topics

<table>
<thead>
<tr>
<th>C. Lists</th>
<th>Sections 6.7, 9.0–9.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>D. Arrays</td>
<td>Sections 6.0, 6.1</td>
</tr>
</tbody>
</table>

### V. STANDARD ALGORITHMS

Standard algorithms serve as examples of good solutions to standard problems. Many are intertwined with standard data structures. These algorithms provide examples for analysis of program efficiency.

#### A. Operations on data structures previously listed

1. **Traversals**
   - Sections 6.0, 6.1, 6.7
2. **Insertions**
   - Sections 6.0, 6.1, 6.7
3. **Deletions**
   - Sections 6.0, 6.1, 6.7

#### B. Searching

1. **Sequential**
   - Section 6.2
2. **Binary**
   - Section 6.2

#### C. Sorting

1. **Selection**
   - Section 6.3
2. **Insertion**
   - Section 6.3
3. **Mergesort**
   - Section 8.3

### VI. COMPUTING IN CONTEXT

An awareness of the ethical and social implications of computing systems is necessary for the study of computer science. These topics need not be covered in detail but should be considered throughout the course.

#### A. System reliability
- Appendix F

#### B. Privacy
- Appendix F

#### C. Legal issues and intellectual property
- Appendix F

#### D. Social and ethical ramifications of computer use
- Appendix F

### Acknowledgments

We are most grateful to the AP* teachers and AP* students from around the world who have provided their feedback on the previous edition of this book. We are pleased to see the depth of the faculty’s concern for their AP* students and the AP* students’ thirst for knowledge. Your comments and questions are always welcome.

Michael Hirsch and Stephanie Sellinger, our editors at Addison-Wesley, went above and beyond the call of duty to ensure that the book met the highest-quality standards. Their support and enthusiasm are greatly appreciated. We are also grateful to Andrea Sheehan and Courtney Marsh with Pearson School for making sure that AP* teachers understand the pedagogical advantages of this text. The devotion that the Addison-Wesley and Pearson folks show to their books is evident in the high-quality results.
The production team for this edition is a group of gifted and hard-working people—miracle workers all. Thanks go to Heather McNally, Joyce Cosentino Wells (for the wonderful cover), and Rose Kernan and the rest of the Nesbitt Graphics group. The quality of the book is due largely to their personal attention, and it is greatly appreciated.

Special thanks also go to the reviewers of this text over the last few editions, as well as the many other instructors and friends who have provided valuable feedback.

They include Neil Ascione—Wayne Valley High School, NJ; Brian Ellis—Manheim Township School District, PA; Kathy Gilbert—Cary-Grove High School, IL; Gregory King—Dublin Coffman High School, OH; Richard Lamb—Cranbrook High School, MI; Deepa Muralidhar—Northview High School, GA; Kim Murphy—Weber High School, UT; G. Lynne Ryan—North Crowley High School, TX; Jenka Guevara, American School Foundation, Mexico City; Leigh Ann Sudol, Fox Lane High School/New York University, NY; Kathleen Weaver, Hillcrest High School, Dallas, TX; John E. Hanna, Teaneck High School, NJ; Brian G. Scarbeau, Lake Highland Preparatory School, FL; Frances Caruso Wolanczyk, Fieldston School, NY; Mark Hanington, Punahou School, Honolulu, HI; Rose M. Hoffman, Catholic Memorial High School, Waukesha, WI; Kathleen Larson, Kingston High School, Kingston, NY; Bob Beck, Villanova University; Paul Gormley, Villanova University; Cathy Helwig, Villanova University; Dan Joyce, Villanova University; Najib Nadi, Villanova University; Beth Taddei, Villanova University; Barbara Zimmerman, Villanova University; John Simms, Marquette University; Marian Manyo, Marquette University.

Most importantly, we thank our families for their support during the busy process of writing, not to mention the other times.
Feature Walkthrough

Key Concepts. Throughout the text, the Key Concept boxes highlight fundamental ideas and important guidelines. These concepts are summarized at the end of each chapter.

Listings. All programming examples are presented in clearly labeled listings, followed by the program output, a sample run, or screen shot display as appropriate. The code is colored to visually distinguish comments and reserved words.

---

disk. When you instruct the computer to execute your program, a copy of the program is brought in from secondary memory and stored in main memory. The CPU reads the individual program instructions from main memory. The CPU then executes the instructions one at a time until the program ends. The data that the instructions use, such as two numbers that will be added together, are also stored in main memory.

The process of executing a program is fundamental to the operation of a computer. All computer systems basically work in the same way.

2.1 Computer Processing

**KEY CONCEPT**
The CPU reads the program instructions from main memory, executing them one at a time until the program ends.

---

**LISTING 2.2**

```java
//********************************************************************
// Roses.java       Author: Lewis/Loftus
//
// Demonstrates the use of escape sequences.
//********************************************************************

public class Roses
{
    public static void main (String[] args)
    {
        System.out.println("Roses are red,\n" +"Violets are blue,\n" +"Sugar is sweet,\n" +"But I have "commitment issues",\n" +"So I’d rather just be friends\n" +"At this point in our relationship.");
    }
}
```

**OUTPUT**

Roses are red,
Violets are blue,
Sugar is sweet,
But I have "commitment issues",
So I’d rather just be friends
At this point in our relationship.
Syntax Diagrams. At appropriate points in the text, syntactic elements of the Java language are discussed in special highlighted sections with diagrams that clearly identify the valid forms for a statement or construct.

Graphics Track. All processing that involves graphics and graphical user interfaces is discussed in one or two sections at the end of each chapter that we collectively refer to as the Graphics Track. This material can be skipped without loss of continuity, or focused on specifically as desired. The material in any Graphics Track section relates to the main topics of the chapter in which it is found. Graphics Track sections are indicated by a light blue border on the edge of the page.

A variable declaration consists of a Type followed by a list of variables. Each variable can be initialized in the declaration to the value of the specified Expression. If the final modifier precedes the declaration, the identifiers are declared as named constants whose
Summary of Key Concepts. The Key Concepts presented throughout a chapter are summarized at the end of the chapter.

Self-Review Questions

2.1 What are the primary concepts that support object-oriented programming?
2.2 Why is an object an example of abstraction?
2.3 What is primitive data? How are primitive data types different from objects?
2.4 What is a string literal?
2.5 What is the difference between the print and println methods?

Answers to Self-Review Questions

2.1 The main elements that support object-oriented programming are objects, classes, encapsulation, and inheritance. An object is defined by a class, which contains methods that define the operations on those objects (the services that they perform). Objects store and manage their own data. Inheritance is a technique in which one class can be created from another.
2.2 An object is abstract because the details of the object are hidden from, and largely unimportant to, the user of the object. Hidden details help us manage the complexity of software.

AP* Case Study. The case study sections apply concepts from the text to the AP* Case Study that students need to study for the AP* Exam. The case study sections can be found at www.aw.com/cssupport under author: Lewis/Loftus/Cocking.

Summary of Key Concepts

- The information we manage in a Java program is either primitive data or objects.
- An abstraction hides details. A good abstraction hides the right details at the right time.
- A variable is a name for a memory location used to hold a value.
- A variable can store only one value of its declared type.
- Java is a strongly typed language. Each variable has a specific type, and we cannot assign a value of one type to a variable of another type.
- Constants are like variables, but they have the same value throughout the program.
- Java has two kinds of numeric values: integers and floating point. The primitive type int is an integer data type and double is a floating point data type.
- Expressions are combinations of one or more operands and the operators used to perform a calculation.
- Java has rules that govern the order in which operators will be evaluated in an expression. These rules are called operator precedence rules.
- Avoid narrowing conversions because they can lose information.
Short Answer
7.1 Draw an inheritance hierarchy containing classes that represent different types of clocks. Show the variables and method names for two of these classes.
7.2 Show another diagram for the hierarchy in Exercise 7.1. Explain why it may be better or worse than the original.
7.3 Draw a class hierarchy for types of teachers at a high school. Show what characteristics would be represented in the various classes of the hierarchy. Explain how polymorphism could play a role in assigning courses to each teacher.

Exercises. The multiple choice, true/false, and short answer exercises at the end of every chapter are designed to develop and test the students’ knowledge of that chapter’s material. These exercises generally do not require the use of a computer.

AP*-Style Questions. These questions are modeled after the multiple choice and free response questions on the AP* Exam. They are designed to help students apply material from each chapter to the types of questions they will encounter on the AP* Exam.

Programming Projects. These problems require the design and implementation of Java programs. They vary widely in level of difficulty.

AP*-Style Multiple Choice
3.1 Consider the following output.
10 9 8 7 5 4 3 2 1
Which of the following loops will produce this output?
(A) for (int i = 0; i < 10; i--)
System.out.print(i + " ");
(B) for (int i = 10; i > 0; i--)
System.out.print(i + " ");
(C) for (int i = 0; i <= 10; i++)
System.out.print(10 – i + " ");
(D) for (int i = 0; i < 10; i++)
System.out.print((10 – i) + " ");
(E) for (int i = 10; i > 0; i--)
System.out.print((10 – i) + " ");

AP*-Style Free Response
5.1 Consider the following incomplete declaration of a Name class.
```java
public class Name { something missing
    private String firstName;
    private String lastName;
    public Name(String firstName, String lastName) {
        firstName = firstName;
        lastName = lastName;
    }
}
```