GLOBAL EDITION

Java Software Solutions

Foundations of Program Design

EIGHTH EDITION

John Lewis • William Loftus

ALWAYS LEARNING

PEARSON

Eighth Edition

java™ SOFTWARE SOLUTIONS

FOUNDATIONS OF PROGRAM DESIGN

Global Edition

JOHN LEWIS
Virginia Tech

•

WILLIAM LOFTUS
Accenture

•

Global Edition contributions by
Mohit Tahiliani
NITK Surathkal

PEARSON

Boston Columbus Indianapolis New York San Francisco Upper Saddle River
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto
Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Editorial Director:	Marcia Horton
Editor-in-Chief:	Michael Hirsch
Editorial Assistant:	Stephanie Sellinger
Vice President, Marketing:	Patrice Jones
Marketing Manager:	Yezan Alayan
Marketing Coordinator:	Kathryn Ferranti
Vice President, Production:	Vince O'Brien
Managing Editor:	Jeff Holcomb
Production Project Manager:	Marilyn Lloyd
Head, Learning Asset Acquisition, Global Edition:	Laura Dent
Acquisitions Editor, Global Edition:	Karthik Subramaniun
Project Editor, Global Edition:	Anuprova Dey Chowdhuri.
Senior Operations Supervisor:	Alan Fischer
Manufacturing Buyer:	Lisa McDowell
Art Director:	Linda Knowles
Cover Designer:	Shree Mohanambal Inbakumar/Lumina Datamatics, Inc.
Image Permission Coordinator:	Rita Wenning
Cover Photograph:	Eugene Sergeev/Shutterstock
Media Editor:	Daniel Sandin
Media Project Manager:	Wanda Rockwell
Full-Service Project Management:	Harleen Chopra, Cenveo® Publisher Services
Composition:	Cenveo Publisher Services
Printer/Binder:	Courier Kendallville
Cover Printer:	Courier Kendallville
Text Font:	Sabon LT Std

Pearson Education Limited
Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

Visit us on the World Wide Web at:
www.pearsonglobaleditions.com

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

Typeset in 8 Sabon LT Std by Cenveo Publishing Services

Printed and bound by Courier Kendallville

The publisher's policy is to use paper manufactured from sustainable forests.

10 9 8 7 6 5 4 3 2 1—DOC—14 13 12 11 10

PEARSON

ISBN 10: 1292018232
ISBN 13: 978-1-29-201823-2 (Print)
ISBN 13: 978-1-29-206977-7 (PDF)

This book is dedicated to our families.
Sharon, Justin, Kayla, Nathan, and Samantha Lewis
and
Veena, Isaac, and Dévi Loftus

This page is intentionally left blank.

Preface

Welcome to the Eighth Edition of *Java Software Solutions: Foundations of Program Design*. We are pleased that this book has served the needs of so many students and faculty over the years. This edition has been tailored further to improve the coverage of topics key to introductory computing.

New to This Edition

The biggest updates to this edition include the following:

- Updated all graphical screen shots.
- Added additional screen shots to show interface options.
- Revised end-of-chapter exercises and programming projects.
- Revised all code for consistent spacing issues.
- Improved discussions of Java translation, text file I/O, and other topics.

Feedback from both instructors and students continues to make it clear that we have hit the mark with the overall vision of the book. The emphasis remains on presenting underlying core concepts in a clear and gradual manner. The Graphics Track sections in each chapter still segregate the coverage of graphics and graphical user interfaces, giving extreme flexibility in how that material gets covered. The casual writing style and entertaining examples still rule the day.

The displays of screen shots for graphics-based programs, including programs with graphical user interfaces, have all been updated. The previous versions were dated in terms of the look-and-feel. We also updated some as needed to improve pedagogy.

On some graphics programs, we added additional screen shots in situations where it was beneficial to see how the program window looks under different situations or window sizes.

We also focused on the end-of-chapter exercises and programming projects in this revision. We added, subtracted, and modified these to provide an appropriate and updated set.

Throughout the book, the examples had become inconsistent in some issues related to the use of white space. We carefully went through each example and code fragment to ensure that a consistent and appropriate style was applied.

Finally, as always, we improved discussions throughout the book, sometimes in minor ways, and a few include significant improvements. In particular, the discussion of Java translation was updated in Chapter 1 and throughout to focus

on term JVM rather than the less helpful term interpreter. The figure related to the translation process was also updated. The text file I/O discussion was also updated, along with its example.

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 a student sees must discuss the writing of multiple classes and methods. A 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.* 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. Students learn how to solve problems as well as how to implement solutions. We introduce and integrate basic software engineering techniques throughout the text. The **Software Failure** vignettes reiterate these lessons by demonstrating the perils of not following these sound practices.
- *Examples.* 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 nongraphics programs. The **VideoNotes** provide additional examples in a live presentation format.
- *Graphics and GUIs.* Graphics can be a great motivator for 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 GUIs. 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 (Introduction) 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 general program development,

as well as object-oriented programming, are discussed. This chapter contains broad introductory material that can be covered while students become familiar with their development environment.

Chapter 2 (Data and Expressions) explores some of the basic types of data used in a Java program and the use of expressions to perform calculations. It discusses the conversion of data from one type to another and how to read input interactively from the user with the help of the standard `Scanner` class.

Chapter 3 (Using Classes and Objects) explores the use of predefined classes and the objects that can be created from them. Classes and objects are used to manipulate character strings, produce random numbers, perform complex calculations, and format output. Enumerated types are also discussed.

Chapter 4 (Writing Classes) explores the basic issues related to writing classes and methods. Topics include instance data, visibility, scope, method parameters, and return types. Encapsulation and constructors are covered as well. Some of the more involved topics are deferred to or revisited in Chapter 6.

Chapter 5 (Conditionals and Loops) covers the use of boolean expressions to make decisions. Then the `if` statement and `while` loop are explored in detail. Once loops are established, the concept of an iterator is introduced and the `Scanner` class is revisited for additional input parsing and the reading of text files. Finally, the `ArrayList` class introduced, which provides the option for managing a large number of objects.

Chapter 6 (More Conditionals and Loops) examines the rest of Java's conditional (`switch`) and loop (`do`, `for`) statements. All related statements for conditionals and loops are discussed, including the enhanced version of the `for` loop. The for-each loop is also used to process iterators and `ArrayList` objects.

Chapter 7 (Object-Oriented Design) reinforces and extends the coverage of issues related to the design of classes. Techniques for identifying the classes and objects needed for a problem and the relationships among them are discussed. This chapter also covers static class members, interfaces, and the design of enumerated type classes. Method design issues and method overloading are also discussed.

Chapter 8 (Arrays) contains extensive coverage of arrays and array processing. The nature of an array as a low-level programming structure is contrasted to the higher-level object management approach. Additional topics include command-line arguments, variable length parameter lists, and multidimensional arrays.

Chapter 9 (Inheritance) covers class derivations and associated concepts such as class hierarchies, overriding, and visibility. Strong emphasis is put on the proper use of inheritance and its role in software design.

Chapter 10 (Polymorphism) explores the concept of binding and how it relates to polymorphism. Then we examine how polymorphic references can be accomplished using either inheritance or interfaces. Sorting is used as an example of polymorphism. Design issues related to polymorphism are examined as well.

Chapter 11 (Exceptions) explores the class hierarchy from the Java standard library used to define exceptions, as well as the ability to define our own exception objects. We also discuss the use of exceptions when dealing with input and output and examine an example that writes a text file.

Chapter 12 (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.

Chapter 13 (Collections) introduces the idea of a collection and its underlying data structure. Abstraction is revisited in this context and the classic data structures are explored. Generic types are introduced as well. This chapter serves as an introduction to a CS2 course.

Supplements

Student Online Resources

These student resources can be accessed at the book's Companion Website, www.pearsonglobaleditions.com/lewis:

- Source Code for all the programs in the text
- Links to Java development environments
- VideoNotes: short step-by-step videos demonstrating how to solve problems from design through coding. VideoNotes allow for self-paced instruction with easy navigation including the ability to select, play, rewind, fast-forward, and stop within each VideoNote exercise. Margin icons in your textbook let you know when a VideoNote video is available for a particular concept or homework problem.

Instructor Resources

The following supplements are available to qualified instructors only. Visit the Pearson Education Instructor Resource Center www.pearsonglobaleditions.com/lewis for information on how to access them:

- Presentation Slides—in PowerPoint.
- Solutions—includes solutions to exercises and programming projects.
- Test Bank with powerful test generator software—includes a wealth of free response, multiple-choice, and true/false type questions.
- Lab Manual—lab exercises are designed to accompany the topic progression in the text.

Features

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.

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. Syntax diagrams for the entire Java language are presented in Appendix L.

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 brown border on the edge of the page.

Summary of Key Concepts. The Key Concepts presented throughout a chapter are summarized at the end of the chapter.

Self-Review Questions and Answers. These short-answer questions review the fundamental ideas and terms established in the preceding section. They are designed to allow students to assess their own basic grasp of the material. The answers to these questions can be found at the end of the book in Appendix N.

Exercises. These intermediate problems require computations, the analysis or writing of code fragments, and a thorough grasp of the chapter content. While the exercises may deal with code, they generally do not require any online activity.

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

VideoNotes. Presented by the author, VideoNotes explain topics visually through informal videos in an easy-to-follow format, giving students the extra help they need to grasp important concepts. Look for this VideoNote icon to see which in-chapter topics and end-of-chapter Programming Projects are available as VideoNotes.

Software Failures. These between-chapter vignettes discuss real-world flaws in software design, encouraging students to adopt sound design practices from the beginning.

Acknowledgments

I am most grateful to the faculty and students from around the world who have provided their feedback on previous editions of this book. I am pleased to see

the depth of the faculty's concern for their students and the students' thirst for knowledge. Your comments and questions are always welcome.

I am particularly thankful for the assistance, insight, and attention to detail of Robert Burton from Brigham Young University. For years, Robert has consistently provided valuable feedback that helps shape and evolve this textbook.

Brian Fraser of Simon Fraser University also has recently provided some excellent feedback that helped clarify some issues in this edition. Such interaction with computing educators is incredibly valuable.

I also want to thank Dan Joyce from Villanova University, who developed the Self-Review questions, ensuring that each relevant topic had enough review material, as well as developing the answers to each.

I continue to be amazed at the talent and effort demonstrated by the team at Pearson. Matt Goldstein, our editor, has amazing insight and commitment. His assistant, Kelsey Loanes, is a source of consistent and helpful support. Marketing Manager Yez Alayan makes sure that instructors understand the pedagogical advantages of the text. The cover was designed by the skilled talents of Joyce Wells. Scott Disanno, Marilyn Lloyd, and Kayla Smith-Tarbox led the production effort. The Addison-Wesley folks were supported by a phenomenal team at Cenveo Publisher Services including Jerilyn Bockorick for the interior design and Harleen Chopra for project management. We thank all of these people for ensuring that this book meets the highest quality standards.

Special thanks go to the following people who provided valuable advice to us about this book via their participation in focus groups, interviews, and reviews. They, as well as many other instructors and friends, have provided valuable feedback. They include:

Elizabeth Adams	James Madison University
Hossein Assadipour	Rutgers University
David Atkins	University of Oregon
Lewis Barnett	University of Richmond
Thomas W. Bennet	Mississippi College
Gian Mario Besana	DePaul University
Hans-Peter Bischof	Rochester Institute of Technology
Don Braffitt	Radford University
Robert Burton	Brigham Young University
John Chandler	Oklahoma State University
Robert Cohen	University of Massachusetts, Boston
Dodi Coreson	Linn Benton Community College
James H. Cross II	Auburn University
Eman El-Sheikh	University of West Florida
Sherif Elfayoumy	University of North Florida

Christopher Eliot	University of Massachusetts, Amherst
Wanda M. Eanes	Macon State College
Stephanie Elzer	Millersville University
Matt Evett	Eastern Michigan University
Marj Feroe	Delaware County Community College, Pennsylvania
John Gauch	University of Kansas
Chris Haynes	Indiana University
James Heliotis	Rochester Institute of Technology
Laurie Hendren	McGill University
Mike Higgs	Austin College
Stephen Hughes	Roanoke College
Daniel Joyce	Villanova University
Saroja Kanchi	Kettering University
Gregory Kapfhammer	Allegheny College
Karen Kluge	Dartmouth College
Jason Levy	University of Hawaii
Peter MacKenzie	McGill University
Jerry Marsh	Oakland University
Blayne Mayfield	Oklahoma State University
Gheorghe Muresan	Rutgers University
Laurie Murphy Pacific	Lutheran University
Dave Musicant	Carleton College
Faye Navabi-Tadayon	Arizona State University
Lawrence Osborne	Lamar University
Barry Pollack	City College of San Francisco
B. Ravikumar	University of Rhode Island
David Riley	University of Wisconsin (La Crosse)
Bob Roos	Allegheny College
Carolyn Rosiene	University of Hartford
Jerry Ross Lane	Community College
Patricia Roth	Southeastern Polytechnic State University
Carolyn Schauble	Colorado State University
Arjit Sengupta	Georgia State University
Bennet Setzer	Kennesaw State University
Vijay Srinivasan	JavaSoft, Sun Microsystems, Inc.
Stuart Steiner	Eastern Washington University
Katherine St. John	Lehman College, CUNY
Alexander Stoytchev	Iowa State University
Ed Timmerman	University of Maryland, University College
Shengru Tu	University of New Orleans
Paul Tymann	Rochester Institute of Technology
John J. Wegis	JavaSoft, Sun Microsystems, Inc.

Ken Williams	North Carolina Agricultural and Technical University
Linda Wilson	Dartmouth College
David Wittenberg	Brandeis University
Wang-Chan Wong	California State University (Dominguez Hills)

Thanks also go to my friends and former colleagues at Villanova University who have provided so much wonderful feedback. They include Bob Beck, Cathy Helwig, Anany Levitin, Najib Nadi, Beth Taddei, and Barbara Zimmerman.

Special thanks go to Pete DePasquale of The College of New Jersey for the design and evolution of the PaintBox project, as well as the original Java Class Library appendix.

Many other people have helped in various ways. They include Ken Arnold, Mike Czepiel, John Loftus, Sebastian Niezgoda, and Saverio Perugini. Our apologies to anyone we may have omitted.

The ACM Special Interest Group on Computer Science Education (SIGCSE) is a tremendous resource. Their conferences provide an opportunity for educators from all levels and all types of schools to share ideas and materials. If you are an educator in any area of computing and are not involved with SIGCSE, you're missing out.

Pearson Education wishes to thank Arup Bhattacharjee, Soumen Mukherjee and Raghavan for reviewing the Global Edition.

Contents

VideoNote

Credits

Cover: © *Eugene Sergeev/Shutterstock*
01-2 Screenshots reprinted with permission from Apple, Inc.
02-SF NASA Earth Observing System
03-3 Java API doucumentation. Used by permission of Oracle Corporate Counsel
04-SFd Susan Van Etten/PhotoEdit
05-SF-b David Joel/Stone/Getty Images
07-SF-c-1 and 07-SF-c-2 National Oceanic and Atmospheric Administration
08-SF-a Matthew McVay/Getty Images
09-SF-e Mario Fourmy/REA/Redux Pictures
Listing 10.13 Robert Frost, The Road Not Taken (1916)
Display Listing 11.13 © MShieldsPhotos/Alamy
M.1 http://docs.oracle.com/javase7/docs/api
UNF02-01, UNF11-01, UNF12-01 John Lewis
© Microsoft Corporation. Used with permission from Microsoft. MICROSOFT AND/OR ITS RESPECTIVE SUPPLIERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE INFORMATION CONTAINED IN THE DOCUMENTS AND RELATED GRAPHICS PUBLISHED AS PART OF THE SERVICES FOR ANY PURPOSE. ALL SUCH DOCUMENTS AND RELATED GRAPHICS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. MICROSOFT AND/OR ITS RESPECTIVE SUPPLIERS HEREBY DISCLAIM ALL WARRANTIES AND CONDITIONS OF MERCHANTABILITY, WHETHER EXPRESS, IMPLIED, OR STATUTORY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL MICROSOFT AND/OR ITS RESPECTIVE SUPPLIERS BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF INFORMATION AVAILABLE FROM THE SERVICES. THE DOCUMENTS AND RELATED GRAPHICS CONTAINED HEREIN COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN. MICROSOFT AND/OR ITS RESPECTIVE SUPPLIERS MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED HEREIN AT ANY TIME. PARTIAL SCREEN SHOTS MAY BE VIEWED IN FULL WITHIN THE SOFTWARE VERSION SPECIFIED.

This page is intentionally left blank.

1 Introduction

CHAPTER OBJECTIVES

- Describe the relationship between hardware and software.
- Define various types of software and how they are used.
- Identify the core hardware components of a computer and explain their roles.
- Explain how the hardware components interact to execute programs and manage data.
- Describe how computers are connected into networks to share information.
- Introduce the Java programming language.
- Describe the steps involved in program compilation and execution.
- Present an overview of object-oriented principles.

This book is about writing well-designed software. To understand software, we must first have a fundamental understanding of its role in a computer system. Hardware and software cooperate in a computer system to accomplish complex tasks. The purpose of various hardware components, and the way those components are connected into networks, are important prerequisites to the study of software development. This chapter first discusses basic computer processing and then begins our exploration of software development by introducing the Java programming language and the principles of object-oriented programming.

1.1 Computer Processing

All computer systems, whether it's a desktop, laptop, tablet, smart phone, gaming console, or a special-purpose device like a car's navigation system, share certain characteristics. The details vary, but they all process data in similar ways. While the majority of this book deals with the development of software, we'll begin with an overview of computer processing to set the context. It's important to establish some fundamental terminology and see how key pieces of a computer system interact.

A computer system is made up of hardware and software. The *hardware* components of a computer system are the physical, tangible pieces that support the computing effort. They include chips, boxes, wires, keyboards, speakers, disks, memory cards, USB flash drives (also called jump drives), cables, plugs, printers, mice, monitors, routers, and so on. If you can physically touch it and it can be considered part of a computer system, then it is computer hardware.

KEY CONCEPT

A computer system consists of hardware and software that work in concert to help us solve problems.

The hardware components of a computer are essentially useless without instructions to tell them what to do. A *program* is a series of instructions that the hardware executes one after another. *Software* consists of programs and the data those programs use. Software is the intangible counterpart to the physical hardware components. Together they form a tool that we can use to help solve problems.

The key hardware components in a computer system are

- central processing unit (CPU)
- input/output (I/O) devices
- main memory
- secondary memory devices

Each of these hardware components is described in detail in the next section. For now, let's simply examine their basic roles. The *central processing unit* (CPU) is the device that executes the individual commands of a program. *Input/output (I/O) devices*, such as the keyboard, mouse, and monitor, allow a human being to interact with the computer.

Programs and data are held in storage devices called memory, which fall into two categories: main memory and secondary memory. *Main memory* is the storage device that holds the software while it is being processed by the CPU. *Secondary memory* devices store software in a relatively permanent manner. The most important secondary memory device of a typical computer system is the hard disk that resides inside the main computer box. A USB flash drive is also an important secondary memory device. A typical USB flash drive cannot store nearly as much information as a hard disk. USB flash drives have the advantage of portability; they can be removed temporarily or moved from computer to computer as needed. Another portable secondary memory device is the compact disc (CD).

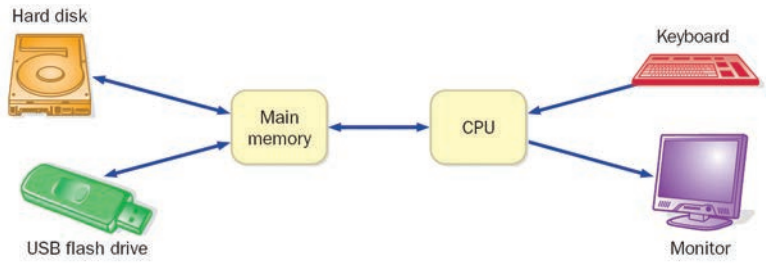

FIGURE 1.1 A simplified view of a computer system

Figure 1.1 shows how information moves among the basic hardware components of a computer. Suppose you have an executable program you wish to run. The program is stored on some secondary memory device, such as a hard 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, also are stored in main memory. They are either brought in from secondary memory or read from an input device such as the keyboard. During execution, the program may display information to an output device such as a monitor.

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

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

Software Categories

Software can be classified into many categories using various criteria. At this point we will simply differentiate between system programs and application programs.

The *operating system* is the core software of a computer. It performs two important functions. First, it provides a *user interface* that allows the user to interact with the machine. Second, the operating system manages computer resources such as the CPU and main memory. It determines when programs are allowed to run, where they are loaded into memory, and how hardware devices communicate. It is the operating system's job to make the computer easy to use and to ensure that it runs efficiently.

KEY CONCEPT
The operating system provides a user interface and manages computer resources.

Several popular operating systems are in use today. The Windows operating system was developed for personal computers by Microsoft, which has captured the lion's share of the operating systems market.

Various versions of the Unix operating system are also quite popular, especially in larger computer systems. A version of Unix called Linux was developed as an open source project, which means that many people contributed to its development and its code is freely available. Because of that, Linux has become a particular favorite among some users. Mac OS X is an operating system used for computing systems developed by Apple Computers.

Operating systems are often specialized for mobile devices such as smart phones and tablets. The iOS operating system from Apple is used on the iPhone, iPad, and iPod Touch. It is similar in functionality and appearance to the desktop Mac OS, but tailored for the smaller devices. Likewise, Windows Phone is the version of the Windows operating system from Microsoft used in their phones. Android is a Linux-based mobile operating system developed by Google and used on many phones.

An *application* (often shortened in conversation to "app") is a generic term for just about any software other than the operating system. Word processors, missile control systems, database managers, Web browsers, and games all can be considered application programs. Each application program has its own user interface that allows the user to interact with that particular program.

The user interface for most modern operating systems and applications is a *graphical user interface* (GUI, pronounced "gooey"), which, as the name implies, makes use of graphical screen elements. Among many others, these elements include

- *windows,* which are used to separate the screen into distinct work areas
- *icons,* which are small images that represent computer resources, such as a file
- *menus, checkboxes, and radio buttons,* which provide the user with selectable options
- *sliders,* which allow the user to select from a range of values
- *buttons,* which can be "pushed" with a mouse click to indicate a user selection

The mouse is the primary input device used with GUIs; thus, GUIs are sometimes called *point-and-click interfaces*. The screen shot in Figure 1.2 shows an example of a GUI.

KEY CONCEPT

As far as the user is concerned, the interface is the program.

The interface to an application or operating system is an important part of the software because it is the only part of the program with which the user interacts directly. To the user, the interface *is* the program. Throughout this book we discuss the design and implementation of graphical user interfaces.

The focus of this book is the development of high-quality application programs. We explore how to design and write software that will perform calculations, make decisions, and present results textually or graphically. We use the Java programming language throughout the text to demonstrate various computing concepts.