GLOBAL EDITION

Java™

An Introduction to Problem Solving & Programming

EIGHTH EDITION

Walter Savitch

Pearson

Digital Resources for Students

Your new textbook provides 12-month access to digital resources that may include VideoNotes (step-by-step video tutorials on programming concepts), source code, web chapters, quizzes, and more. Refer to the preface in the textbook for a detailed list of resources.

Follow the instructions below to register for the Companion Website for Walter Savitch's *Java™: An Introduction to Problem Solving and Programming,* Eighth Edition, Global Edition.

1. Go to www.pearsonglobaleditions.com/Savitch
2. Enter the title of your textbook or browse by author name.
3. Click Companion Website.
4. Click Register and follow the on-screen instructions to create a login name and password.

ISSIPS-PRANK-BURRY-ENDUE-GABBY-TOUSE

Use the login name and password you created during registration to start using the online resources that accompany your textbook.

IMPORTANT:

This prepaid subscription does not include access to Pearson MyLab Programming, which is available at www.myprogramminglab.com for purchase.

This access code can only be used once. This subscription is valid for 12 months upon activation and is not transferable. If the access code has already been revealed it may no longer be valid.

For technical support go to https://support.pearson.com/getsupport

Java™

An Introduction to Problem Solving & Programming

Eighth edition

Global edition

Java™

Eighth edition

Global edition

An Introduction to
Problem Solving & Programming

Walter Savitch
University of California, San Diego

Contributor
Kenrick Mock
University of Alaska Anchorage

Pearson

330 Hudson Street, New York, NY 10013

Senior Vice President Courseware Portfolio Management:	Marcia J. Horton
Director, Portfolio Management: Engineering, Computer Science & Global Editions:	Julian Partridge
Portfolio Manager:	Matt Goldstein
Portfolio Management Assistant:	Kristy Alaura
Acquisitions Editor, Global Edition:	Sourabh Maheshwari
Assistant Project Editor, Global Edition:	Aurko Mitra
Field Marketing Manager:	Demetrius Hall
Product Marketing Manager:	Yvonne Vannatta
Managing Producer, ECS and Math:	Scott Disanno
Content Producer:	Sandra L. Rodriguez
Media Production Manager, Global Edition:	Vikram Kumar
Senior Manufacturing Controller, Global Edition:	Caterina Pellegrino
Cover Designer:	Lumina Datamatics, Inc.
Cover Photo:	Racheal Grazias/Shutterstock

Pearson Education Limited
KAO Two
KAO Park
Harlow
CM17 9NA
United Kingdom

and Associated Companies throughout the world

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

ISBN 10: 1-292-24747-9
ISBN 13: 978-1-292-24747-2

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

10 9 8 7 6 5 4 3 2 1

Printed and bound by Vivar in Malaysia

Typeset by iEnergizer Aptara®, Ltd.

Preface for Instructors

Welcome to the eighth edition of *Java: An Introduction to Problem Solving & Programming*. This book is designed for a first course in programming and computer science. It covers programming techniques, as well as the basics of the Java programming language. It is suitable for courses as short as one quarter or as long as a full academic year. No previous programming experience is required, nor is any mathematics, other than a little high school algebra. The book can also be used for a course designed to teach Java to students who have already had another programming course, in which case the first few chapters can be assigned as outside reading.

Changes in This Edition

The following list highlights how this eighth edition differs from the seventh edition:

- Correction of errors and edits for readability.
- The material on Java applets has been removed from the printed text but is available as an online chapter.
- With the exception of `JOptionPane` the graphics supplements have changed from Swing to JavaFX. The Swing chapters are available online. The JavaFX material introduces drawing, layout, event handling, and common UI controls.
- Examples of event-driven programming with the event handler in a separate class, the main application class, an anonymous inner class, and using lambda functions.
- Introduction to the Timeline and Scene Builder.
- Five new VideoNotes for a total of seventy seven VideoNotes. These VideoNotes walk students through the process of both problem solving and coding to help reinforce key programming concepts. An icon appears in the margin of the book when a VideoNote is available regarding the topic covered in the text.
- Ten new/revised Programming Projects.

Latest Java Coverage

All of the code in this book has been tested using Oracle's Java SE Development Kit (JDK), version 8. Any imported classes are standard and in the Java Class Library that is part of Java. No additional classes or specialized libraries are needed.

Flexibility

If you are an instructor, this book adapts to the way you teach, rather than making you adapt to the book. It does not tightly prescribe the sequence in which your course must cover topics. You can easily change the order in which you teach many chapters and sections. The particulars involved in rearranging material are explained in the dependency chart that follows this preface and in more detail in the "Prerequisites" section at the start of each chapter.

Early Graphics

Graphics supplement sections in each of the chapters. This gives you the option of covering graphics and GUI programming from the start of your course. The graphics supplement sections emphasize GUIs built using `JavaFX`. Any time after Chapter 8, you can move on to the supplemental chapters on GUI programming using Swing (Chapters 13 through 15), which are now on the Web. Alternatively, you can continue through Chapter 10 with a mix of graphics and more traditional programming. Instructors who prefer to postpone the coverage of graphics can postpone or skip the graphics supplement sections.

Coverage of Problem-Solving and Programming Techniques

This book is designed to teach students basic problem-solving and programming techniques and is not simply a book about Java syntax. It contains numerous case studies, programming examples, and programming tips. In addition, many sections explain important problem-solving and programming techniques, such as loop design techniques, debugging techniques, style techniques, abstract data types, and basic object-oriented programming techniques, including UML, event-driven programming, and generic programming using type parameters.

Early Introduction to Classes

Any course that really teaches Java must teach classes early, since everything in Java involves classes. A Java program is a class. The data type for strings of characters is a class. Even the behavior of the equals operator (==) depends on whether it is comparing objects from classes or simpler data items. Classes cannot be avoided, except by means of absurdly long and complicated "magic formulas." This book introduces classes fairly early. Some exposure to using classes is given in Chapters 1 and 2. Chapter 5 covers how to define classes. All of the basic information about classes, including inheritance, is presented by the end of Chapter 8 (even if you omit Chapter 7). However, some topics regarding classes, including inheritance, can be postponed until later in the course.

Although this book introduces classes early, it does not neglect traditional programming techniques, such as top-down design and loop design techniques. These older topics may no longer be glamorous, but they are information that all beginning students need.

Generic Programming

Students are introduced to type parameters when they cover lists in Chapter 12. The class `ArrayList` is presented as an example of how to use a class that has a type parameter. Students are then shown how to define their own classes that include a type parameter.

Language Details and Sample Code

This book teaches programming technique, rather than simply the Java language. However, neither students nor instructors would be satisfied with an introductory programming course that did not also teach the programming language. Until you calm students' fears about language details, it is often impossible to focus their attention on bigger issues. For this reason, the book gives complete explanations of Java language features and lots of sample code. Programs are presented in their entirety, along with sample input and output. In many cases, in addition to the complete examples in the text, extra complete examples are available over the Internet.

Self-Test Questions

Self-test questions are spread throughout each chapter. These questions have a wide range of difficulty levels. Some require only a one-word answer, whereas others require the reader to write an entire, nontrivial program. Complete answers for all the self-test questions, including those requiring full programs, are given at the end of each chapter.

Exercises and Programming Projects

Completely new exercises appear at the end of each chapter. Since only you, and not your students, will have access to their answers, these exercises are suitable for homework. Some could be expanded into programming projects. However, each chapter also contains other programming projects, several of which are new to this edition.

Support Material

The following support materials are available on the Internet at www .pearsonglobaleditions.com/Savitch:

For instructors only:

- Solutions to most exercises and programming projects
- PowerPoint slides

Instructors should click on the registration link and follow instructions to receive a password. If you encounter any problems, please contact your local Pearson Sales Representative.

For students:

- Source code for programs in the book and for extra examples
- VideoNotes: video solutions to programming examples and exercises.

Visit www.pearsonglobaleditions.com/Savitch to access the student resources.

Online Practice and Assessment with Pearson MyLab Programming

Pearson MyLab Programming helps students fully grasp the logic, semantics, and syntax of programming. Through practice exercises and immediate, personalized feedback, MyLab Programming improves the programming competence of beginning students who often struggle with the basic concepts and paradigms of popular high-level programming languages.

A self-study and homework tool, a MyLab Programming course consists of hundreds of small practice problems organized around the structure of this textbook. For students, the system automatically detects errors in the logic and syntax of their code submissions and offers targeted hints that enable students to figure out what went wrong—and why. For instructors, a comprehensive gradebook tracks correct and incorrect answers and stores the code inputted by students for review.

MyLab Programming is offered to users of this book in partnership with Turing's Craft, the makers of the CodeLab interactive programming exercise system. For a full demonstration, to see feedback from instructors and students, or to get started using MyLab Programming in your course, visit www.myprogramminglab.com.

VideoNote

VideoNotes

VideoNotes are designed for teaching students key programming concepts and techniques. These short step-by-step videos demonstrate how to solve problems from design through coding. VideoNotes allow for self-placed 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.

Contact Us

Your comments, suggestions, questions, and corrections are always welcome. Please e-mail them to savitch.programming.java@gmail.com.

Preface for Students

This book is designed to teach you the Java programming language and, even more importantly, to teach you basic programming techniques. It requires no previous programming experience and no mathematics other than some simple high school algebra. However, to get the full benefit of the book, you should have Java available on your computer, so that you can practice with the examples and techniques given. The latest version of Java is preferable.

If You Have Programmed Before

You need no previous programming experience to use this book. It was designed for beginners. If you happen to have had experience with some other programming language, do not assume that Java is the same as the programming language(s) you are accustomed to using. All languages are different, and the differences, even if small, are large enough to give you problems. Browse the first four chapters, reading at least the Recap portions. By the time you reach Chapter 5, it would be best to read the entire chapter.

If you have programmed before in either C or C++, the transition to Java can be both comfortable and troublesome. At first glance, Java may seem almost the same as C or C++. However, Java is very different from these languages, and you need to be aware of the differences. Appendix 6 compares Java and C++ to help you see what the differences are.

Obtaining a Copy of Java

Appendix 1 provides links to sites for downloading Java compilers and programming environments. For beginners, we recommend Oracle's Java JDK for your Java compiler and related software and TextPad or DrJava as a simple editor environment for writing Java code. When downloading the Java JDK, be sure to obtain the latest version available.

Support Materials for Students

- Source code for programs in the book and for extra examples
- Student lab manual
- VideoNotes: video solutions to programming examples and exercises.

Visit www.pearsonglobaleditions.com/Savitch to access the student resources.

Learning Aids

Each chapter contains several features to help you learn the material:

- The opening overview includes a brief table of contents, chapter objectives and prerequisites, and a paragraph or two about what you will study.
- Recaps concisely summarize major aspects of Java syntax and other important concepts.
- FAQs, or "frequently asked questions," answer questions that other students have asked.
- Remembers highlight important ideas you should keep in mind.
- Programming Tips suggest ways to improve your programming skills.
- Gotchas identify potential mistakes you could make—and should avoid—while programming.
- Asides provide short commentaries on relevant issues.
- Self-Test Questions test your knowledge throughout, with answers given at the end of each chapter. One of the best ways to practice what you are learning is to do the self-test questions *before* you look at the answers.
- A summary of important concepts appears at the end of each chapter.

Online Practice with Pearson MyLab Programming

A self-study and practice tool, a MyLab Programming course consists of hundreds of small practice problems organized around the structure of this textbook. The system automatically detects errors in the logic and syntax of your code submissions and offers targeted hints that enable you to figure out what went wrong—and why. Visit www.myprogramminglab.com for more information.

VideoNotes

VideoNote

These short step-by-step videos demonstrate how to solve problems from design through coding. VideoNotes allow for self-placed 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.

This Text Is Also a Reference Book

In addition to using this book as a textbook, you can and should use it as a reference. When you need to check a point that you have forgotten or that you hear mentioned by somebody but have not yet learned yourself, just look in the index. Many index entries give a page number for a "recap." Turn to that page. It will contain a short, highlighted entry giving all the essential points on that topic. You can do this to check details of the Java language as well as details on programming techniques.

Recap sections in every chapter give you a quick summary of the main points in that chapter. Also, a summary of important concepts appears at the end of each chapter. You can use these features to review the chapter or to check details of the Java language.

This page intentionally left blank

Acknowledgments

We thank the many people who have made this eighth edition possible, including everyone who has contributed to the first seven editions. We begin by recognizing and thanking the people involved in the development of this new edition. The comments and suggestions of the following reviewers were invaluable and are greatly appreciated. In alphabetical order, they are:

Christopher Crick—*Oklahoma State University*
Christopher Plaue—*University of Georgia*
Frank Moore—*University of Alaska Anchorage*
Frank Witmer—*University of Alaska Anchorage*
Greg Gagne—*Westminster College*
Helen Hu—*Westminster College*
Paul Bladek—*Edmonds Community College, Washington*
Paul LaFollette—*Temple University*
Pei Wang—*Temple University*
Richard Cassoni—*Palomar College*
Walter Pistone—*Palomar College*

Many other reviewers took the time to read drafts of earlier editions of the book. Their advice continues to benefit this new edition. Thank you once again to:

Adel Elmaghraby—*University of Louisville*
Alan Saleski—*Loyola University Chicago*
Anthony Larrain—*DePaul University*
Arijit Sengupta—*Raj Soin College of Business, Wright State University*
Asa Ben-Hur—*Colorado State University*
Ashraful A. Chowdhury—*Georgia Perimeter College*
Billie Goldstein—*Temple University*
Blayne Mayfield—*Oklahoma State University*
Boyd Trolinger—*Butte College*
Charles Hoot—*Oklahoma City University*
Chris Hoffmann—*University of Massachusetts, Amherst*
Dan Adrian German—*Indiana University*
Dennis Brylow—*Marquette University*
Dolly Samson—*Hawaii Pacific University*
Donald E. Smith—*Rutgers University*
Drew McDermott—*Yale University*
Ed Gellenbeck—*Central Washington University*
Faye Tadayon-Navabi—*Arizona State University*
Gerald Baumgartner—*Louisiana State University*
Gerald H. Meyer—*LaGuardia Community College*
Gobi Gopinath—*Suffolk County Community College*

Gopal Gupta—*University of Texas, Dallas*
H. E. Dunsmore—*Purdue University, Lafayette*
Helen H. Hu—*Westminster College*
Howard Straubing—*Boston College*
James Roberts—*Carnegie Mellon University*
Jim Buffenbarger—*Boise State University*
Joan Boone—*University of North Carolina at Chapel Hill*
John Motil—*California State University, Northridge*
Ken Slonneger—*University of Iowa*
Laird Dornan—*Sun Microsystems, Inc.*
Le Gruenwald—*University of Oklahoma*
Lily Hou—*Carnegie Mellon University*
Liuba Shrira—*Brandeis University*
Martin Chetlen—*Moorpark College*
Mary Elaine Califf—*Illinois State University*
Michael Clancy—*University of California, Berkeley*
Michael Litman—*Western Illinois University*
Michael Long—*California State University*
Michael Olan—*Richard Stockton College of New Jersey*
Michal Young—*University of Oregon*
Michele Kleckner—*Elon University*
Nan C. Schaller—*Rochester Institute of Technology*
Peter Spoerri—*Fairfield University*
Ping-Chu Chu—*Fayetteville State University*
Prasun Dewan—*University of North Carolina, Chapel Hill*
Ricci Heishman—*North Virginia Community College*
Richard A. Johnson—*Missouri State University*
Richard Ord—*University of California, San Diego*
Richard Whitehouse—*Arizona State University*
Robert Herrmann—*Sun Microsystems, Inc., Java Soft*
Robert Holloway—*University of Wisconsin, Madison*
Robert P. Burton—*Brigham Young University*
Rob Kelly—*State University of New York at Stony Brook*
Ryan Shoemaker—*Sun Microsystems, Inc.*
Stan Kwasny—*Washington University*
Stephen F. Weiss—*University of North Carolina, Chapel Hill*
Steven Cater—*Kettering University*
Subramanian Vijayarangam—*University of Massachusetts, Lowell*
Tammy VanDeGrift—*University of Portland*
Thomas Cortina—*Carnegie Mellon University*
Thomas VanDrunen—*Wheaton College*
Y. Annie Liu—*State University of New York at Stony Brook*

We thank Frank Carrano for his revision of the fifth edition of this textbook. Last but not least, we thank the many students in classes at the University of California, San Diego (UCSD), who were kind enough to help correct preliminary versions of this text, as well as the instructors who class-tested these drafts. In particular, we extend a special thanks to Carole McNamee of California State University, Sacramento, and to Paul Kube of UCSD. These student comments and the detailed feedback and class testing of earlier editions of the book were a tremendous help in shaping the final book.

W. S.
K. M.

Acknowledgments for the Global Edition

Pearson would like to thank and acknowledge the following people for their contributions to this Global Edition.

Contributors

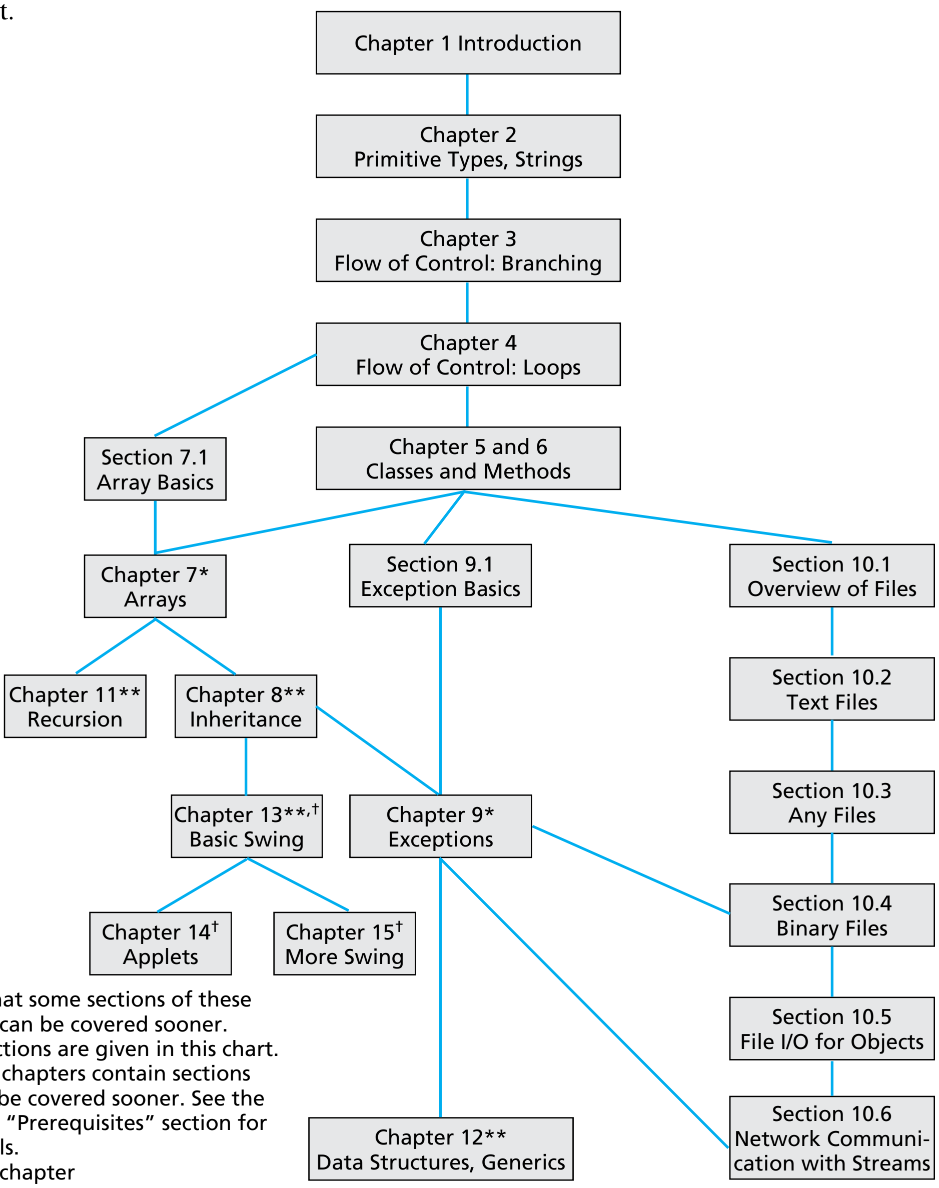
Komal Arora

Reviewers

Arup Bhattacharya—*RCC Institute of Technology*
Ajay Mittal—*University Institute of Engineering and Technology*
Khyat Sharma

Dependency Chart

This chart shows the prerequisites for the chapters in the book. If there is a line between two boxes, the material in the higher box should be covered before the material in the lower box. Minor variations to this chart are discussed in the "Prerequisites" section at the start of each chapter. These variations usually provide more, rather than less, flexibility than what is shown on the chart.



Chapter 1 Introduction
Chapter 2 Primitive Types, Strings
Chapter 3 Flow of Control: Branching
Chapter 4 Flow of Control: Loops
Section 7.1 Array Basics
Chapter 5 and 6 Classes and Methods
Chapter 7* Arrays
Section 9.1 Exception Basics
Section 10.1 Overview of Files
Chapter 11** Recursion
Chapter 8** Inheritance
Section 10.2 Text Files
Chapter 13**,† Basic Swing
Chapter 9* Exceptions
Section 10.3 Any Files
Chapter 14† Applets
Chapter 15† More Swing
Section 10.4 Binary Files
Section 10.5 File I/O for Objects
Chapter 12** Data Structures, Generics
Section 10.6 Network Communication with Streams

* Note that some sections of these chapters can be covered sooner. Those sections are given in this chart.
** These chapters contain sections that can be covered sooner. See the chapter's "Prerequisites" section for full details.
† Online chapter

Features of This Text

Recaps

Summarize Java syntax and other important concepts.

Remembers

Highlight important ideas that students should keep in mind.

RECAP Bytes and Memory Locations

A computer's main memory is divided into numbered units called bytes. The number of a byte is called its address. Each byte can hold eight binary digits, or bits, each
data that is too large to fi
adjacent bytes. These adja
memory location whose a
bytes.

REMEMBER Syntactic Variables

When you see something in this book like *Type*, *Variable_1,* or *Variable_2* used to describe Java syntax, these words do not literally appear in your Java code. They are **syntactic variables,** which are a kind of blank that you fill in with something from the category that they describe. For example, *Type* can be replaced by `int`, `double`, `char`, or any other type name. *Variable_1* and *Variable_2* can each be replaced by any variable name.

Programming Tips

Give students helpful advice about programming in Java.

■ PROGRAMMING TIP Initialize Variables

A variable that has been declared, but that has not yet been given a value by an assignment statement (or in some other way), is said to be **uninitialized.** If the variable is a variable of a class type, it literally has no value. If the variable has a primitive type, it likely has some default value. However, your program will be clearer if you explicitly give the variable a value, even if you are simply reassigning the default value. (The exact details on default values have been known to change and should not be counted on.)

One easy way to ensure that you do not have an uninitialized variable is to initialize it within the declaration. Simply combine the declaration and an assignment statement, as in the following examples:

```
int count = 0;
double taxRate = 0.075;
char grade = 'A';
int balance = 1000, newBalance;
```

Note that you can initialize some variables and not initialize others in a declaration.

Sometimes the compiler may complain that you have failed to initialize a variable. In most cases, that will indeed be true. Occasionally, though, the compiler is mistaken in giving this advice. However, the compiler will not compile your program until you convince it that the variable in question is initialized. To make the compiler happy, initialize the variable when you declare it, even if the variable will be given another value before it is used for anything. In such cases, you cannot argue with the compiler. ■

Gotchas

Identify potential mistakes in programming that students might make and should avoid.

GOTCHA Hidden Errors

Just because your program compiles and runs without any errors and even produces reasonable-looking output does not mean that your program is correct. You should always run your program with some test data that gives predictable output. To do this, choose some data for which you can compute the correct results, either by using pencil and paper, by looking up the answer, or by some other means. Even this testing does not guarantee that your program is correct, but the more testing you do, the more confidence you can have in your program. ■

FAQs

Provide students answers to frequently asked questions within the context of the chapter.

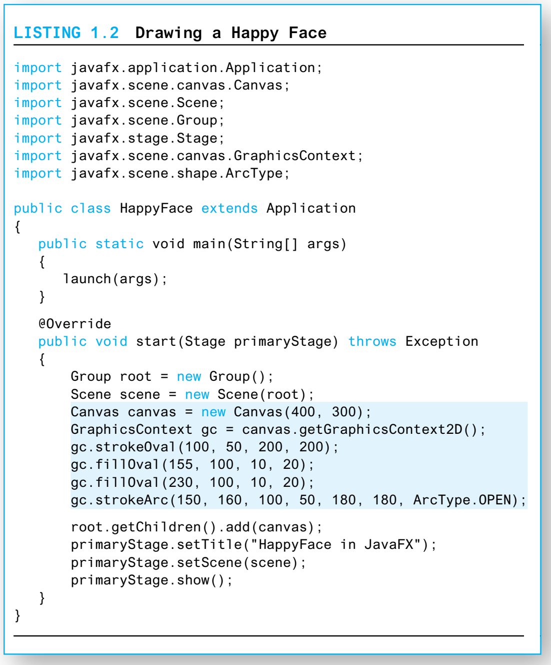
FAQ[1] Why just 0s and 1s?

Computers use 0s and 1s because it is easy to make an electrical device that has only two stable states. However, when you are programming, you normally need not be concerned about the encoding of data as 0s and 1s. You can program as if the computer directly stored numbers, letters, or strings of characters in memory.

There is nothing special about calling the states *zero* and *one*. We could just as well use any two names, such as *A* and *B* or *true* and *false*. The important thing is that the underlying physical device has two stable states, such as on and off or high voltage and low voltage. Calling these two states *zero* and *one* is simply a convention, but it's one that is almost universally followed.

Listings

Show students complete programs with sample output.

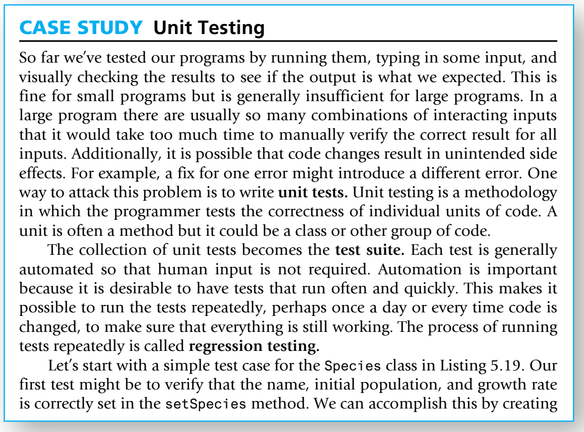


LISTING 1.2 Drawing a Happy Face

```
import javafx.application.Application;
import javafx.scene.canvas.Canvas;
import javafx.scene.Scene;
import javafx.scene.Group;
import javafx.stage.Stage;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.shape.ArcType;

public class HappyFace extends Application
{
   public static void main(String[] args)
   {
      launch(args);
   }

   @Override
   public void start(Stage primaryStage) throws Exception
   {
       Group root = new Group();
       Scene scene = new Scene(root);
       Canvas canvas = new Canvas(400, 300);
       GraphicsContext gc = canvas.getGraphicsContext2D();
       gc.strokeOval(100, 50, 200, 200);
       gc.fillOval(155, 100, 10, 20);
       gc.fillOval(230, 100, 10, 20);
       gc.strokeArc(150, 160, 100, 50, 180, 180, ArcType.OPEN);

       root.getChildren().add(canvas);
       primaryStage.setTitle("HappyFace in JavaFX");
       primaryStage.setScene(scene);
       primaryStage.show();
   }
}
```

Case Studies

Take students from problem statement to algorithm development to Java code.



CASE STUDY Unit Testing

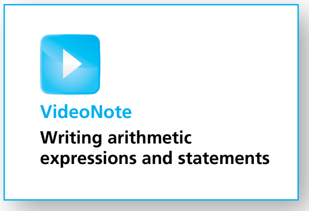
So far we've tested our programs by running them, typing in some input, and visually checking the results to see if the output is what we expected. This is fine for small programs but is generally insufficient for large programs. In a large program there are usually so many combinations of interacting inputs that it would take too much time to manually verify the correct result for all inputs. Additionally, it is possible that code changes result in unintended side effects. For example, a fix for one error might introduce a different error. One way to attack this problem is to write **unit tests.** Unit testing is a methodology in which the programmer tests the correctness of individual units of code. A unit is often a method but it could be a class or other group of code.

The collection of unit tests becomes the **test suite.** Each test is generally automated so that human input is not required. Automation is important because it is desirable to have tests that run often and quickly. This makes it possible to run the tests repeatedly, perhaps once a day or every time code is changed, to make sure that everything is still working. The process of running tests repeatedly is called **regression testing.**

Let's start with a simple test case for the `Species` class in Listing 5.19. Our first test might be to verify that the name, initial population, and growth rate is correctly set in the `setSpecies` method. We can accomplish this by creating

VideoNotes

Step-by-step video solutions to programming examples and homework exercises.



VideoNote
Writing arithmetic expressions and statements

Programming Examples

Provide more examples of Java programs that solve specific problems.

PROGRAMMING EXAMPLE Nested Loops

The body of a loop can contain any sort of statements. In particular, you can have a loop statement within the body of a larger loop statement. For example, the program in Listing 4.4 uses a `while` loop to compute the average of a list of nonnegative scores. The program asks the user to enter all the scores followed by a negative sentinel value to mark the end of the data. This `while` loop is placed inside a `do-while` loop so that the user can repeat the entire process for another exam, and another, until the user wishes to end the program.

Self-Test Questions

Provide students with the opportunity to practice skills learned in the chapter. Answers at the end of each chapter give immediate feedback.

SELF-TEST QUESTIONS

28. Given the class `Species` as defined in Listing 5.19, why does the following program cause an error message?

```
    public class SpeciesEqualsDemo
    {
    public static void main(String[] args)
    {
        Species s1, s2; s1.
        setSpecies("Klingon ox", 10, 15);
        s2.setSpecies("Klingon ox", 10, 15);
        if (s1 == s2)
            System.out.println("Match with ==.");
        else
            System.out.println("Do Not match with ==.")
    }
}
```

29. After correcting the program in the previous question, what output does the program produce?

30. What is the biggest difference between a parameter of a primitive type and a parameter of a class type?

31. Given the class `Species`, as defined in Listing 5.19, and the class

Asides

Give short commentary on relevant topics.

ASIDE Use of the Terms *Parameter* and *Argument*

Our use of the terms *parameter* and *argument* is consistent with common usage. We use *parameter* to describe the definition of the data type and variable inside the header of a method and *argument* to describe items passed into a method when it is invoked. However, people often use these terms interchangeably. Some people use the term *parameter* both for what we call a *formal parameter* and for what we call an *argument*. Other people use the term *argument* both for what we call a *formal parameter* and for what we call an *argument*. When you see the term *parameter* or *argument* in other books, you must figure out its exact meaning from the context.

Brief Contents

The following chapters, along with an index to their contents, are on the book's Website:

Contents