David A. Poplawski

OBJECTS *have Class!*

An Introduction to Programming with Java

OBJECTS *have Class!*

{OBJECTS} *have Class!*

An Introduction to Programming with Java

David A. Poplawski
Michigan Technological University

McGraw Hill

Boston Burr Ridge, IL Dubuque, IA Madison, WI New York San Francisco St. Louis
Bangkok Bogotá Caracas Kuala Lumpur Lisbon London Madrid Mexico City
Milan Montreal New Delhi Santiago Seoul Singapore Sydney Taipei Toronto

McGraw-Hill Higher Education

A Division of The **McGraw-Hill** *Companies*

OBJECTS HAVE CLASS! AN INTRODUCTION TO PROGRAMMING WITH JAVA

Published by McGraw-Hill, a business unit of The McGraw-Hill Companies, Inc., 1221 Avenue of the Americas, New York, NY 10020. Copyright © 2002 by The McGraw-Hill Companies, Inc. All rights reserved. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written consent of The McGraw-Hill Companies, Inc., including, but not limited to, in any network or other electronic storage or transmission, or broadcast for distance learning.

Some ancillaries, including electronic and print components, may not be available to customers outside the United States.

This book is printed on acid-free paper.

1 2 3 4 5 6 7 8 9 0 QPF/QPF 0 9 8 7 6 5 4 3 2 1

ISBN 0–07–242340–4
ISBN 0–07–112258–3 (ISE)

General manager: *Thomas E. Casson*
Publisher: *Elizabeth A. Jones*
Developmental editor: *Melinda Dougharty*
Executive marketing manager: *John Wannemacher*
Senior project manager: *Gloria G. Schiesl*
Lead production supervisor: *Sandra Hahn*
Coordinator of freelance design: *Michelle D. Whitaker*
Freelance cover/interior designer: *Jamie E. O'Neal*
Cover image: *Paul Biddle/The Image Bank*
Supplement producer: *Brenda A. Ernzen*
Media technology senior producer: *Phillip Meek*
Compositor: *Interactive Composition Corporation*
Typeface: *10/12 Times Roman*
Printer: *Quebecor World Fairfield, PA*

Library of Congress Cataloging-in-Publication Data

Poplawski, David A.
Objects have class! : an introduction to programming with Java / David A. Poplawski. — 1st ed.
p. cm.
ISBN 0–07–242340–4 — ISBN 0–07–112258–3 (ISE)
Includes index.
1. Java (Computer program language). 2. Object-oriented programming. I. Title.

QA76.73.J38 P67 2002
005.13'3—dc21 2001026619
CIP

INTERNATIONAL EDITION ISBN 0–07–112258–3
Copyright © 2002. Exclusive rights by The McGraw-Hill Companies, Inc., for manufacture and export. This book cannot be re-exported from the country to which it is sold by McGraw-Hill. The International Edition is not available in North America.

www.mhhe.com

DEDICATION

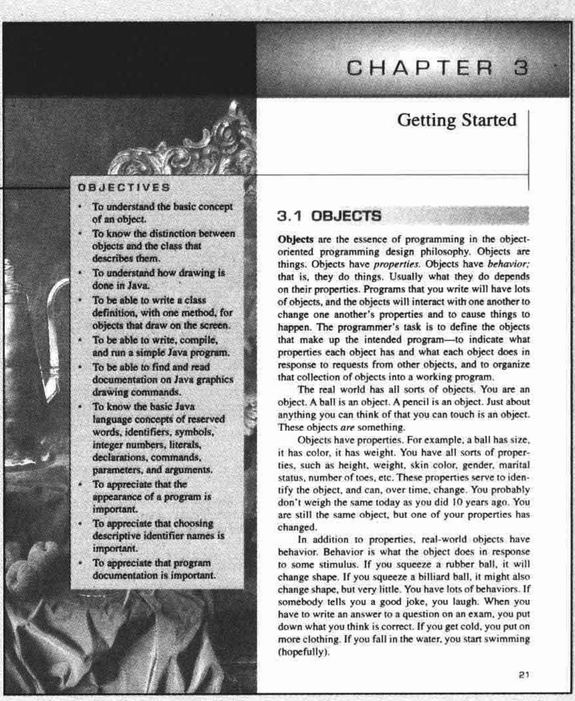
To Dick Rudzinski, who took the time to point me in the right direction a long time ago.

Guided TOUR

Here are some features to help you study programming with Java!

Chapter Objectives

Each chapter opens with a bulleted list of objectives that lets students know what they can expect to learn from the chapter.



CHAPTER 3

Getting Started

OBJECTIVES

- To understand the basic concept of an object.
- To know the distinction between objects and the class that describes them.
- To understand how drawing is done in Java.
- To be able to write a class definition, with one method, for objects that draw on the screen.
- To be able to write, compile, and run a simple Java program.
- To be able to find and read documentation on Java graphics drawing commands.
- To know the basic Java language concepts of reserved words, identifiers, symbols, integer numbers, literals, declarations, commands, parameters, and arguments.
- To appreciate that the appearance of a program is important.
- To appreciate that choosing descriptive identifier names is important.
- To appreciate that program documentation is important.

3.1 OBJECTS

Objects are the essence of programming in the object-oriented programming design philosophy. Objects are things. Objects have *properties.* Objects have *behavior;* that is, they do things. Usually what they do depends on their properties. Programs that you write will have lots of objects, and the objects will interact with one another to change one another's properties and to cause things to happen. The programmer's task is to define the objects that make up the intended program—to indicate what properties each object has and what each object does in response to requests from other objects, and to organize that collection of objects into a working program.

The real world has all sorts of objects. You are an object. A ball is an object. A pencil is an object. Just about anything you can think of that you can touch is an object. These objects *are* something.

Objects have properties. For example, a ball has size, it has color, it has weight. You have all sorts of properties, such as height, weight, skin color, gender, marital status, number of toes, etc. These properties serve to identify the object, and can, over time, change. You probably don't weigh the same today as you did 10 years ago. You are still the same object, but one of your properties has changed.

In addition to properties, real-world objects have behavior. Behavior is what the object does in response to some stimulus. If you squeeze a rubber ball, it will change shape. If you squeeze a billiard ball, it might also change shape, but very little. You have lots of behaviors. If somebody tells you a good joke, you laugh. When you have to write an answer to a question on an exam, you put down what you think is correct. If you get cold, you put on more clothing. If you fall in the water, you start swimming (hopefully).

21

{Definition of Terms}

A function is a method that returns a value. This value is the result of computing the function, using the argument(s) sent to it when it was called.

{New Concepts}

Declaring a method name public makes it accessible from any method in any class. Declaring it private makes it accessible only from methods in the same class.

{New Concepts}

A method executes in the context of an object and as a result has access to all the instance variables of that object.

{Good Ideas}

Don't overspecify an abstraction. Define just those properties and behaviors that are necessary and no more. Adding extra information or constraints will only limit choices for implementations.

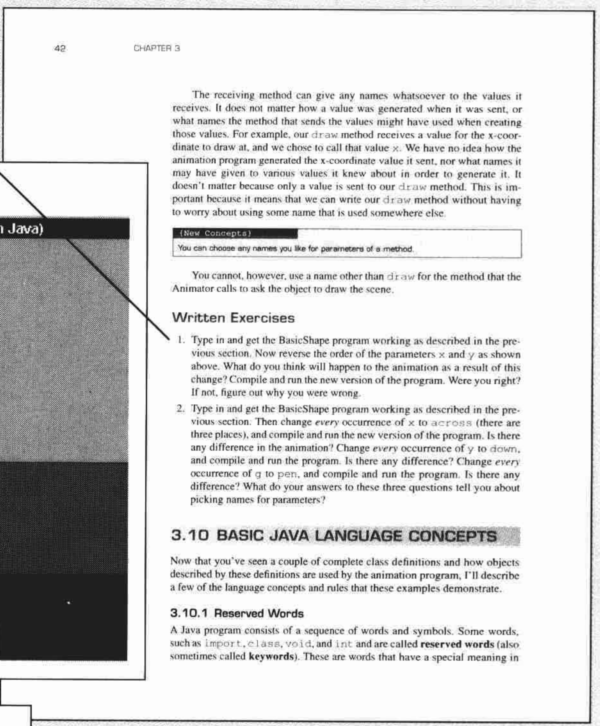
Points of Emphasis

Point-of-Emphasis boxes call attention to especially important definitions, concepts, and ideas, and make them stand out for later reference.

- **Definitions** give clear, concise meanings for important programming terms.
- **Concepts** explain succinctly how particular aspects of programs work.
- **Good Ideas** present guidelines that experience has shown are worth following.

Written Exercises

Written exercises, which appear at the end of almost every chapter section, allow students to test and reinforce their understanding of important material.



42 CHAPTER 3

The receiving method can give any names whatsoever to the values it receives. It does not matter how a value was generated when it was sent, or what names the method that sends the values might have used when creating those values. For example, our draw method receives a value for the x-coordinate to draw at, and we chose to call that value x. We have no idea how the animation program generated the x-coordinate value it sent, nor what names it may have given to various values it knew about in order to generate it. It doesn't matter because only a value is sent to our draw method. This is important because it means that we can write our draw method without having to worry about using some name that is used somewhere else.

(New Concepts)

You can choose any names you like for parameters of a method.

You cannot, however, use a name other than draw for the method that the Animator calls to ask the object to draw the scene.

Written Exercises

1. Type in and get the BasicShape program working as described in the previous section. Now reverse the order of the parameters x and y as shown above. What do you think will happen to the animation as a result of this change? Compile and run the new version of the program. Were you right? If not, figure out why you were wrong.
2. Type in and get the BasicShape program working as described in the previous section. Then change *every* occurrence of x to across (there are three places), and compile and run the new version of the program. Is there any difference in the animation? Change *every* occurrence of y to down, and compile and run the program. Is there any difference? Change *every* occurrence of g to pen, and compile and run the program. Is there any difference? What do your answers to these three questions tell you about picking names for parameters?

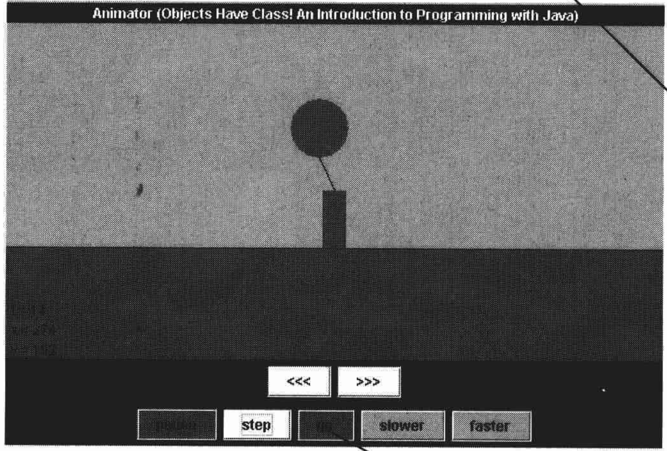
3.10 BASIC JAVA LANGUAGE CONCEPTS

Now that you've seen a couple of complete class definitions and how objects described by these definitions are used by the animation program, I'll describe a few of the language concepts and rules that these examples demonstrate.

3.10.1 Reserved Words

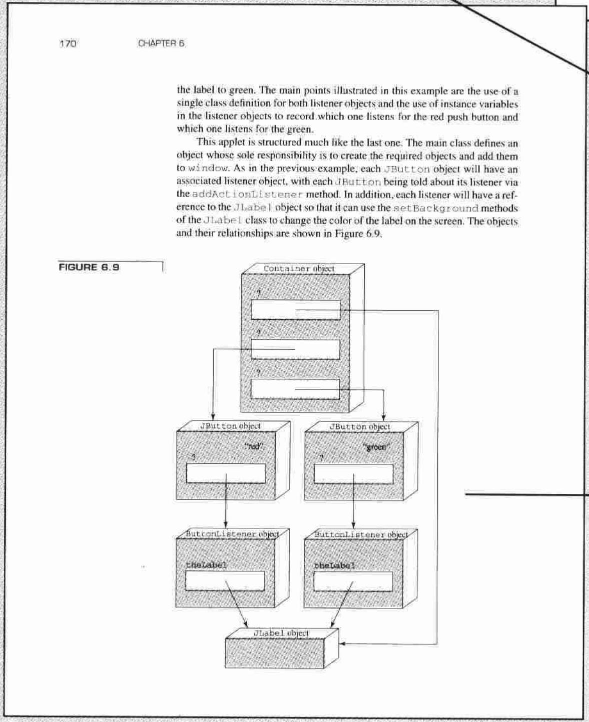
A Java program consists of a sequence of words and symbols. Some words, such as import, class, void, and int and are called **reserved words** (also sometimes called **keywords**). These are words that have a special meaning in

FIGURE 6.1


Animator (Objects Have Class! An Introduction to Programming with Java)
<<<
>>>
step
slower
faster

Animator

The Animator supports the definition and creation of interacting objects, one of the core concepts of object-oriented programming. From the very beginning, it allows students to write simple, short programs that come alive through animation.



170 CHAPTER 6

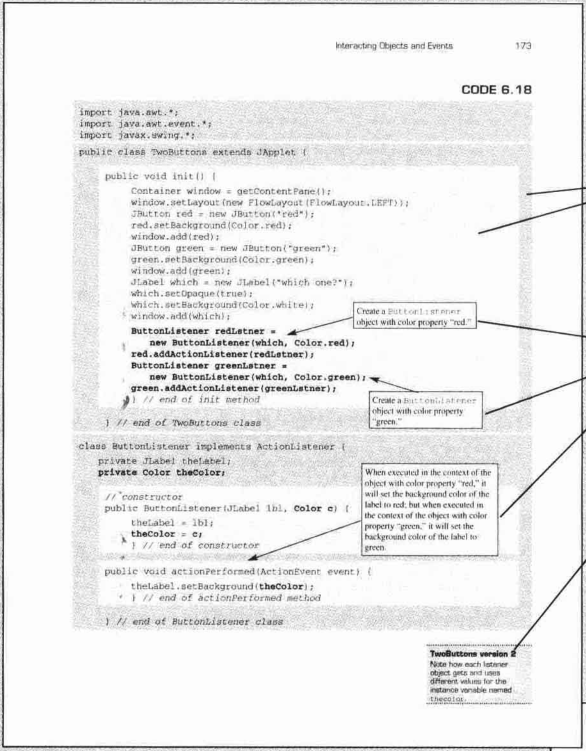
the label to green. The main points illustrated in this example are the use of a single class definition for both listener objects and the use of instance variables in the listener objects to record which one listens for the red push button and which one listens for the green.

This applet is structured much like the last one. The main class defines an object whose sole responsibility is to create the required objects and add them to window. As in the previous example, each JButton object will have an associated listener object, with each JButton being told about its listener via the addActionListener method. In addition, each listener will have a reference to the JLabel object so that it can use the setBackground methods of the JLabel class to change the color of the label on the screen. The objects and their relationships are shown in Figure 6.9.

FIGURE 6.9

Object Diagrams

Diagrams of objects, their instance variables, and their relationships are used to help students visualize how their collections of class definitions result in the creation of objects when the program executes.



CODE 6.18

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TwoButtons extends JApplet {

    public void init() {
        Container window = getContentPane();
        window.setLayout(new FlowLayout(FlowLayout.LEFT));
        JButton red = new JButton("red");
        red.setBackground(Color.red);
        window.add(red);
        JButton green = new JButton("green");
        green.setBackground(Color.green);
        window.add(green);
        JLabel which = new JLabel("which one?");
        which.setOpaque(true);
        Which.setBackground(Color.white);
        window.add(which);
        ButtonListener redLstner =
           new ButtonListener(which, Color.red);
        red.addActionListener(redLstner);
        ButtonListener greenLstner =
           new ButtonListener(which, Color.green);
        green.addActionListener(greenLstner);
    } // end of init method

} // end of TwoButtons class
class ButtonListener implements ActionListener {
    private JLabel theLabel;
    private Color theColor;

    // constructor
    public ButtonListener(JLabel lbl, Color c) {
        theLabel = lbl;
        theColor = c;
    } // end of constructor

    public void actionPerformed(ActionEvent event) {
        theLabel.setBackground(theColor);
    } // end of actionPerformed method

} // end of ButtonListener class
```

Create a ButtonListener object with color property "red."

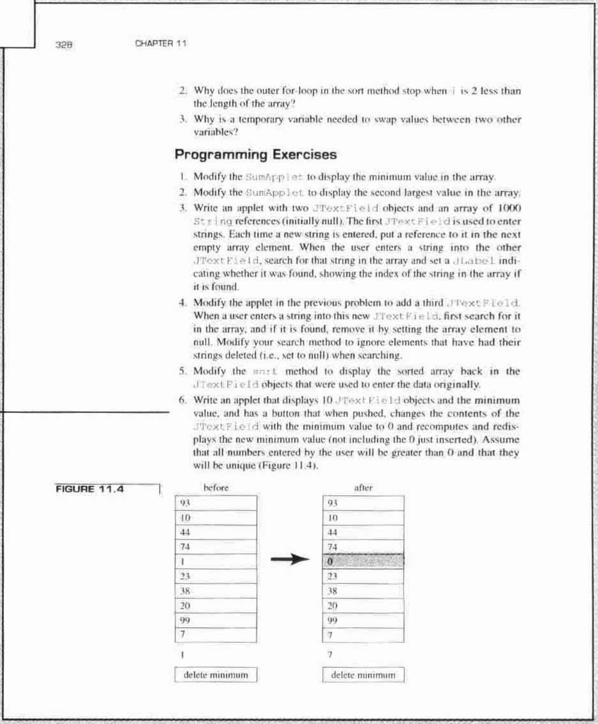
Create a ButtonListener object with color property "green."

When executed in the context of the object with color property "red," it will set the background color of the label to red; but when executed in the context of the object with color property "green," it will set the background color of the label to green.

TwoButtons version 2
Note how each listener object gets and uses different values for the instance variable named theColor.

Innovative Code Formatting

- **Intuitive Shading:** Sections of code are formatted so that classes are highlighted in one intensity and methods in another. This makes identifying and understanding the different elements of code easier for the novice programmer.
- **Author Comments:** Author tips and explanations provide detailed, even line-by-line, instruction on writing correct, workable code.
- **Program identification boxes:** Marginal boxes throughout the text name and describe each new complete program as it is introduced.



2. Why does the outer for-loop in the sort method stop when i is 2 less than the length of the array?
3. Why is a temporary variable needed to swap values between two other variables?

Programming Exercises

1. Modify the SumApplet to display the minimum value in the array.
2. Modify the SumApplet to display the second largest value in the array.
3. Write an applet with two JTextField objects and an array of 1000 String references (initially null). The first JTextField is used to enter strings. Each time a new string is entered, put a reference to it in the next empty array element. When the user enters a string into the other JTextField, search for that string in the array and set a JLabel indicating whether it was found, showing the index of the string in the array if it is found.
4. Modify the applet in the previous problem to add a third JTextField. When a user enters a string into this new JTextField, first search for it in the array, and if it is found, remove it by setting the array element to null. Modify your search method to ignore elements that have had their strings deleted (i.e., set to null) when searching.
5. Modify the sort method to display the sorted array back in the JTextField objects that were used to enter the data originally.
6. Write an applet that displays 10 JTextField objects and the minimum value, and has a button that when pushed, changes the contents of the JTextField with the minimum value to 0 and recomputes and redisplays the new minimum value (not including the 0 just inserted). Assume that all numbers entered by the user will be greater than 0 and that they will be unique (Figure 11.4).

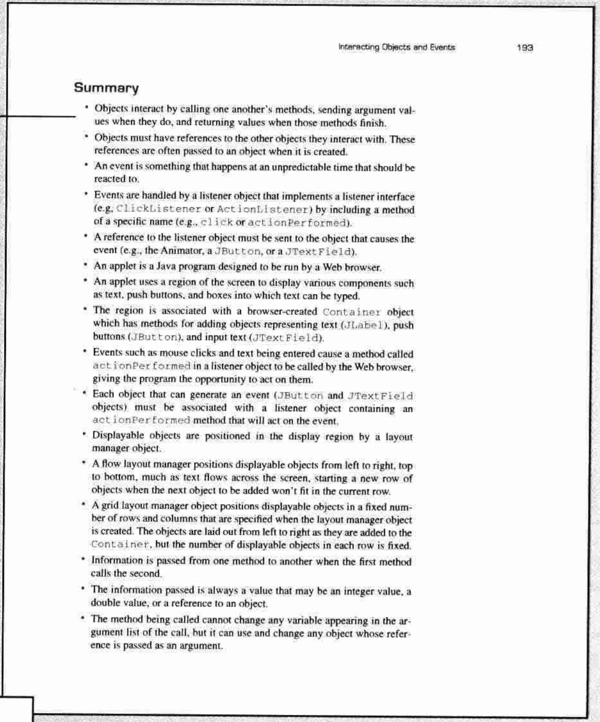
FIGURE 11.4

Programming Exercises

These exercises give the student an opportunity to try out the programming skills that have been introduced in each chapter.

Chapter Summaries

These allow students to review key points and synthesize newly acquired information.



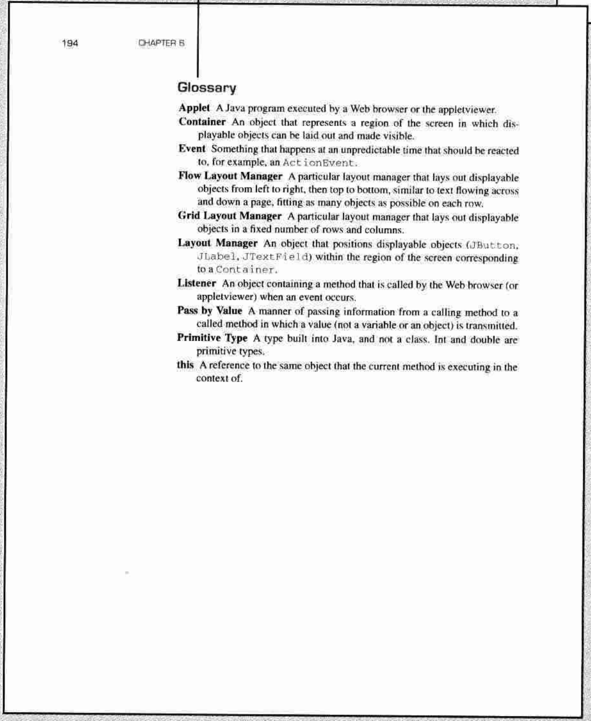
Interacting Objects and Events 193

Summary

- Objects interact by calling one another's methods, sending argument values when they do, and returning values when those methods finish.
- Objects must have references to the other objects they interact with. These references are often passed to an object when it is created.
- An event is something that happens at an unpredictable time that should be reacted to.
- Events are handled by a listener object that implements a listener interface (e.g. `ClickListener` or `ActionListener`) by including a method of a specific name (e.g., `click` or `actionPerformed`).
- A reference to the listener object must be sent to the object that causes the event (e.g., the Animator, a `JButton`, or a `JTextField`).
- An applet is a Java program designed to be run by a Web browser.
- An applet uses a region of the screen to display various components such as text, push buttons, and boxes into which text can be typed.
- The region is associated with a browser-created `Container` object which has methods for adding objects representing text (`JLabel`), push buttons (`JButton`), and input text (`JTextField`).
- Events such as mouse clicks and text being entered cause a method called `actionPerformed` in a listener object to be called by the Web browser, giving the program the opportunity to act on them.
- Each object that can generate an event (`JButton` and `JTextField` objects) must be associated with a listener object containing an `actionPerformed` method that will act on the event.
- Displayable objects are positioned in the display region by a layout manager object.
- A flow layout manager positions displayable objects from left to right, top to bottom, much as text flows across the screen, starting a new row of objects when the next object to be added won't fit in the current row.
- A grid layout manager object positions displayable objects in a fixed number of rows and columns that are specified when the layout manager object is created. The objects are laid out from left to right as they are added to the `Container`, but the number of displayable objects in each row is fixed.
- Information is passed from one method to another when the first method calls the second.
- The information passed is always a value that may be an integer value, a double value, or a reference to an object.
- The method being called cannot change any variable appearing in the argument list of the call, but it can use and change any object whose reference is passed as an argument.

Chapter Glossaries

Key terms are listed and briefly defined at the end of each chapter.



194 CHAPTER 6

Glossary

Applet A Java program executed by a Web browser or the appletviewer.

Container An object that represents a region of the screen in which displayable objects can be laid out and made visible.

Event Something that happens at an unpredictable time that should be reacted to, for example, an `ActionEvent`.

Flow Layout Manager A particular layout manager that lays out displayable objects from left to right, then top to bottom, similar to text flowing across and down a page, fitting as many objects as possible on each row.

Grid Layout Manager A particular layout manager that lays out displayable objects in a fixed number of rows and columns.

Layout Manager An object that positions displayable objects (`JButton`, `JLabel`, `JTextField`) within the region of the screen corresponding to a `Container`.

Listener An object containing a method that is called by the Web browser (or appletviewer) when an event occurs.

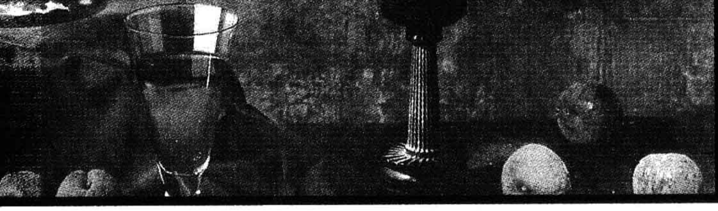
Pass by Value A manner of passing information from a calling method to a called method in which a value (not a variable or an object) is transmitted.

Primitive Type A type built into Java, and not a class. Int and double are primitive types.

this A reference to the same object that the current method is executing in the context of.

Supplements

- A CD-ROM packaged with the book contains all source code and the Animator.
- The Online Learning Center includes solutions, course notes, lab preparations and lab programming assignments, homework programming assignments, quiz material, links to professional resources, source code, and a downloadable version of the Animator.



PREFACE

Welcome! *Objects Have Class! An Introduction to Programming with Java* is not your typical introductory programming text. Programming is fun! And that is the first thing beginning students should learn. In this book they get to do object-oriented programming from the outset. Essential concepts are introduced through animations, allowing students to form a visual image of objects and their interactions.

Our students today come to computer science from a culture that is media-rich. *Objects Have Class!* offers them access to the powers of programming by giving them a fun-to-use, easy-to-grasp programming tool called the Animator. The Animator and the Objects Have Class! approach make both teaching and learning introductory programming a highly intuitive, more dynamic enterprise than ever before.

SPECIAL FEATURES

1. Object Orientation

Students define and use their own classes and objects from the beginning. These essential concepts of object-oriented programming begin in Chapter 3 and then are continually reinforced throughout the book. In fact, traditional programming concepts such as conditionals don't even appear until Chapter 7. Virtually every example and every programming exercise in the text involve multiple objects. Simple inheritance is introduced early (Chapter 9), but only in a very limited way by extending standard Java classes (e.g., JPanel and MouseAdapter). Inheritance is revisited more completely in Chapter 15, when students are ready to design and use their own simple class hierarchies.

2. The Animator

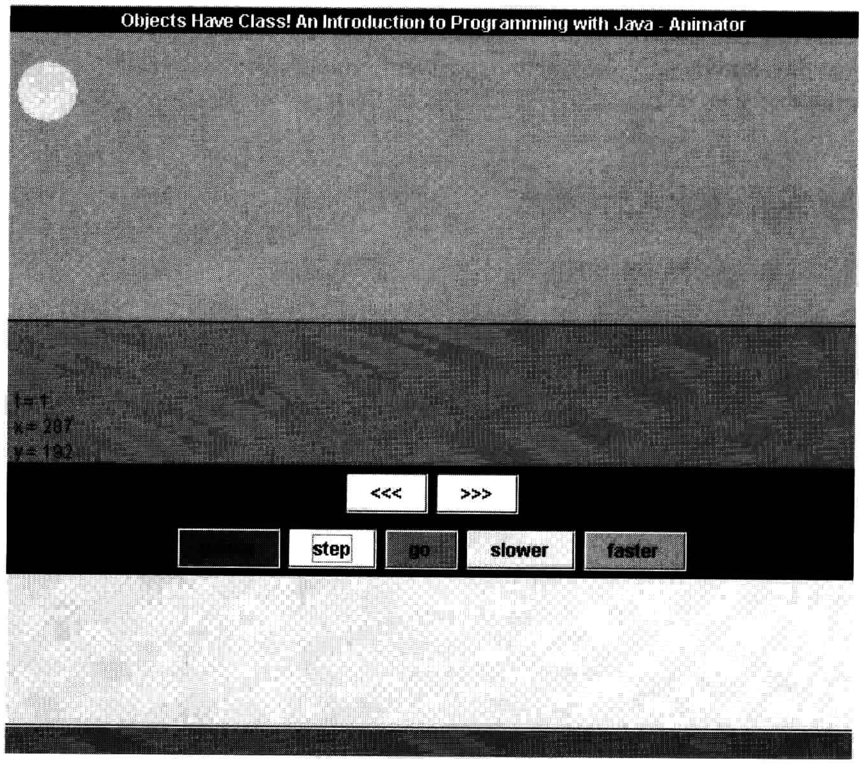
With the Animator students use graphics instead of textual output in the early chapters to write simple, short programs that come alive through animation. Drawing simple graphical shapes (lines, circles, rectangles, etc.) in color is intuitive to most students. Hence graphics is used instead of textual output in the early chapters. The Animator provides an environment in which objects defined by the students draw initially simple, then increasingly more complex animations in which the objects interact to vary their appearance and movement.

For example, the very first example program a student sees is this one:

```
import java.awt*;

public class Sun extends Animator {

    public void draw(Graphics g) {
        g.setColor(Color.yellow);
        g.fillOval(10,20,40,40);
        }

}
```

This simple class definition produces an object that draws a fixed, yellow sun in the upper left corner of the animation scene area (Figure 1).

FIGURE 1


Objects Have Class! An Introduction to Programming with Java - Animator
t = 1
x = 207
y = 132
<<<
>>>
step
go
slower
faster

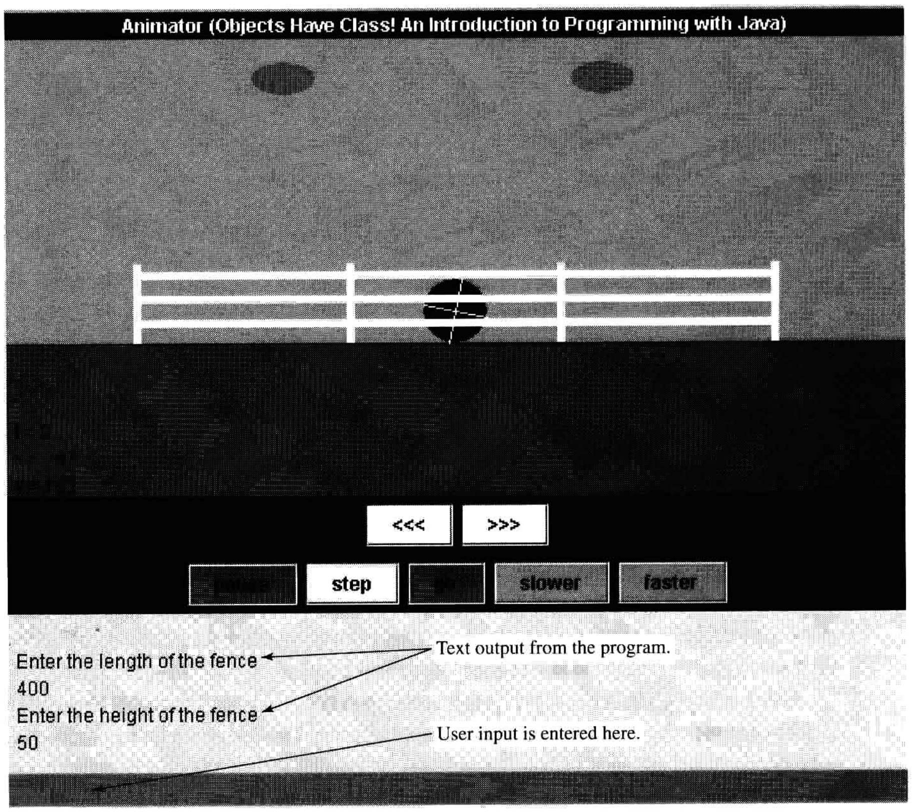
This simple program introduces the student to the concepts of classes verses objects, methods that encode the behavior of objects, and calling methods defined for another object (Graphics) to draw the yellow circle.

The key point to the animations is that each object's behavior includes drawing something in the animation scene. This one-to-one correspondence between an object and a figure in the scene makes it easy for students to *see* the effect of various code structures, and it gives them immediate, visual feedback when they make mistakes.

Successive examples involve drawing increasingly complex figures. This motivates the use of parameter values and the writing of nontrivial arithmetic expressions, conditional structures, and simple loops. Having Java objects directly represent shapes drawn in the scene also emphasizes that objects have properties (state) that must be saved, modified, and used to affect what each object draws as the animation progresses.

Programs with multiple objects, several of which are defined by the same class, again emphasize the difference between class and object and also familiarize students with creating their own objects. The example in Figure 2 (a wheel rolling behind a fence and clouds drifting through the sky) contains several student-defined objects, some moving, others not. This example also shows the simple mechanism provided by the Animator that allows programs to read and write simple textual information similar to traditional text-based, blocking input and output commands. Students need not deal with Java's awkward input and output classes until much later, making it possible for them to display debugging information and to get user input to direct various aspects of an animation.

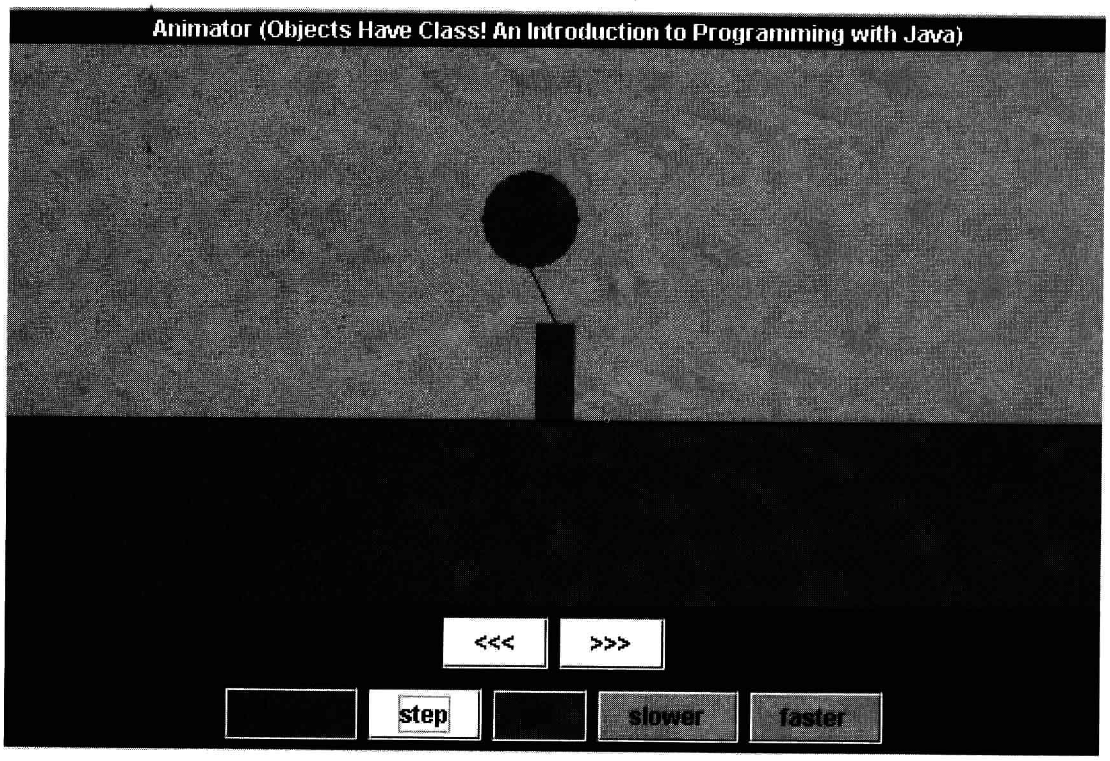
FIGURE 2


Animator (Objects Have Class! An Introduction to Programming with Java)
<<<
>>>
step
slower
faster
Text output from the program.
User input is entered here.

```
Enter the length of the fence
400
Enter the height of the fence
50
```

Eventually animations are defined in which several objects are created that interact with one another to exchange information via method calls, parameter passing, and return values. The example in Figure 3 consists of a fixed "person" (the tall rectangle), a balloon that can be moved from left to right (the circle), and a string (the line) between them. Each is represented by a separate object. The string object uses its references to the person and balloon objects to get their positions in the scene and then draw the line between them appropriately. Object interactions such as these are simple and provide visually based feedback to the student on whether the objects are interacting correctly. Understanding object interactions such as these is essential to understanding object-oriented programs, and the animation environment simplifies it for the student.

FIGURE 3


Animator (Objects Have Class! An Introduction to Programming with Java)
<<<
>>>
step
slower
faster

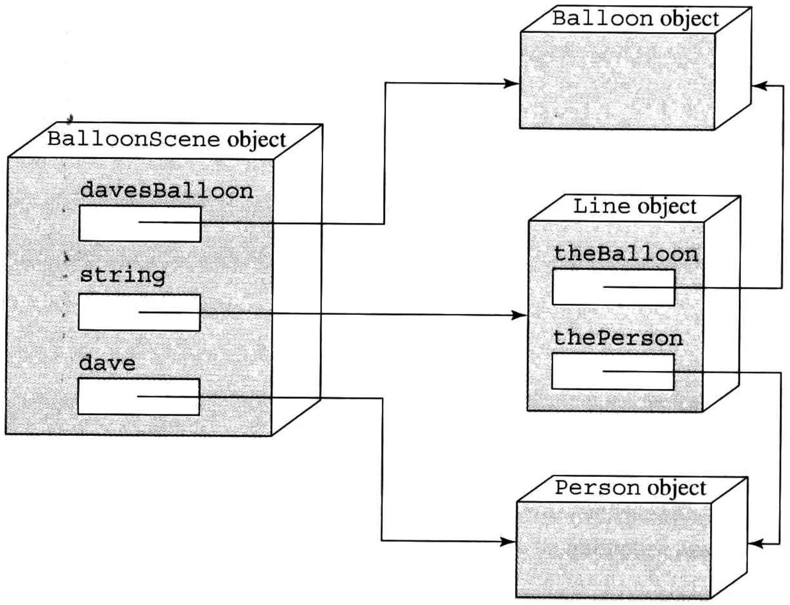
The Animator also allows the student to vary the speed of the animation, or even stop it and then single-step to see what is drawn on each refresh of the scene. When combined with the simple textual output commands, this provides a simple but fairly robust debugging environment.

Every complete animation program in the text can be retrieved from the text's associated CD and/or website and executed. Doing so will help students see exactly what an animation does and how the code makes it happen.

3. Object Diagrams

Diagrams of objects, their instance variables, and their relationships are used to help students visualize how their collection of class definitions results in the creation of objects when their program executes. Figure 4 shows the objects that define the previous balloon animation, showing in particular that the object that draws the line in the scene has references to the balloon and person objects so that it can find out their positions and thereby draw the string between them correctly.

FIGURE 4



The use of such diagrams in the early design stages of a program is emphasized. In fact the student is encouraged to draw similar diagrams before writing any code.

4. Applets and Applications

All programming examples, problems, and solutions in the first several chapters use the Animator. By the middle of Chapter 6, students will have gained enough understanding and experience with multiclass programs that define multiple, interacting objects, including some that react to asynchronous events, that students can begin writing applets with event-driven buttons and text fields. Applets require the use of these same concepts, only in a different setting than animations, and hence the use of object-oriented designs in the text is continued. Applets are preferred over applications because applets can be linked to the student's own Web pages and viewed via Web browsers, increasing motivation for students who can "show off" their work to their friends and relatives. The Animator runs as an applet, too, so even the student's animations can be put on display.

Application programs are presented in Chapter 12, but designing them with graphical interfaces is emphasized. The transition to application programs with GUIs from applets is simple and natural, requiring only one new class (JFrame). Application programs then serve as the basis on which programs using Java's textual, binary, and serialized object input/output classes are built.

5. Points of Emphasis

At selected points certain aspects of the discussion in the narrative become important enough to emphasize. These points of emphasis come in three types:

- *Definitions* give clear, concise meanings to important programming terms.

{Definitions of Terms}

A *class* is a description of one or more objects. It describes the kinds of properties the objects have, but not their particular values, and it describes the behaviors of the objects.

- *Concepts* explain succinctly how certain aspects of programs work.

{New Concepts}

Objects interact by calling one another's methods. They send information via arguments and can get information returned.

- *Good ideas* are not rules, but guidelines that experience has shown are worth following.

{Good Ideas}

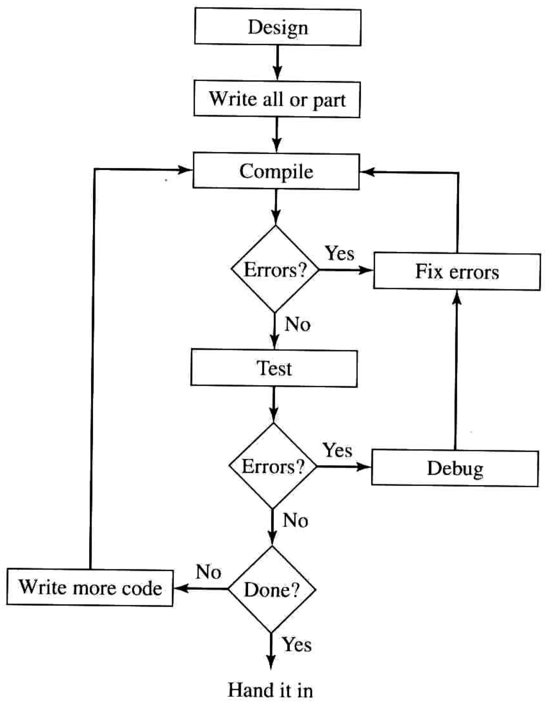
Incremental and unit testing helps isolate possible errors to small sections of code, thereby making debugging easier and more efficient.

Because they stand out from the text, they clearly point out what is important. They also provide a good study guide, as more extensive explanation and examples can be found nearby in the text.

6. Program Development

The basic program development process (Figure 5) is presented in Chapter 2 and reinforced throughout the book, including an emphasis on designing before coding, incremental development, and good debugging and testing techniques. Debugging hints are given in almost every chapter, and students are warned about commonly made errors. The thought processes that are involved in developing various parts of programs, and not just final products, are presented (e.g., Section 10.5, "An Approach to Writing Loops").

FIGURE 5



Chapter 8 is about testing, including black-and-white (or clear) box tests, unit tests, incremental testing, and drivers and stubs. Succeeding chapters apply these testing techniques in new contexts as they arise. Manual execution is presented as a method for understanding how a code segment or entire program works (or doesn't), especially when students are learning to write loops, to manipulate arrays and linked lists, and to do recursion.

7. End-of-Chapter Glossaries

Using the correct terminology to refer to programming concepts is important to understanding. Each chapter concludes with a glossary in which all new terms covered in the chapter are defined and summarized.

8. Java and Swing

This text is not about the Java language. It is about learning how to program using an object-oriented, graphical approach. Java is a great language for this purpose as it supports object-oriented design; has built-in graphical support in the form of the AWT and Swing libraries; has a clean, simple design that lends itself to use by beginning programmers; and is just plain fun to use. The Swing extensions to the window tool kit are used to keep pace with the rapid evolution of Java and its supporting libraries.

CHAPTER OVERVIEW

Chapter 1, **"Computers, Programs, and Java,"** provides background information for students with little or no computer experience. For most students this will simply be a review, although the glossary terms will make many loosely defined terms more precise.

Chapter 2, **"Writing Programs,"** defines the design, code, test, and debug cycle for developing correctly working programs, and it describes the kinds of software tools used in each step. It also makes the first strong case for spending time designing a program before jumping headlong into coding.

Chapter 3, **"Getting Started,"** guides the student through the first steps in writing simple programs consisting of a single class definition with a single method. Each object defined by one of these simple classes is used with the Animator to draw a simple shape, either static and moving, within the animation scene. The emphasis in this chapter is on distinguishing classes versus objects, and how a method describes one behavior of an object. The basic language concepts of symbols, identifiers, literals, declarations, simple parameter passing, and comments are defined, and the need for neat-looking, well-documented code is demonstrated.

Chapter 4, **"Variables, Expressions, and Assignment,"** contains fairly formal definitions of variables of the int, double, and String types as well as the evaluation of expressions using the basic arithmetic operators, type conversions, and functions from Java's Math library. All these are used to define more complex shapes and movement within the animation scene. Debugging techniques based on displaying variable and expression values are introduced to help the students identify and isolate errors in expressions.

Chapter 5, **"Defining and Creating Multiple Objects,"** is the first exposure to programs with multiple, student-defined objects, including several different objects defined by the same class. Instance variables are introduced and used to contain property values for the objects and to reference other objects. At this point object interactions are quite simple, but parameter passing is revisited in depth since this is the first time students write calls to methods that they write themselves. This is also the first time name scoping becomes an issue, and the chapter begins the discussion of the scope of instance, parameter, and local variables. Object diagrams are also introduced to help students visualize the relationships between objects.

Chapter 6, **"Interacting Objects and Events,"** begins describing how objects interact via method calls and return values. The concepts are motivated and demonstrated via animations in which objects representing shapes can exchange information about one another (e.g., position), leading to some interesting animated scenes. Passing object references is covered for the first time. The clicking of the mouse within the scene is presented as a first, simple introduction to the concept of an asynchronous event, and techniques for reacting to the event within an animation object are shown. The chapter concludes by introducing simple Java applets with labels, buttons, text fields, and the ability to