Preface

We have made a number of improvements in this third edition of the book, but the main objectives remain the same. This book is intended as an introductory text on object-oriented programming, suitable for use in a one-semester CS1 course, and assumes no prior programming experience from the students. We only assume basic computer skills and some background in algebra and trigonometry to solve certain chapter exercises. Those who already have experience in traditional non-object-oriented programming languages such as C, BASIC, and others also can use this book as an introduction to object-oriented programming, graphical user interface, and event-driven programming. The two main objectives of this book are to teach

- object-oriented programming, and
- the foundations of real-world programming.

Object-orientation has become an important paradigm in all fields of computer science, and it is important to teach object-oriented programming from the first programming course. Teaching object-oriented programming is more than teaching the syntax and semantics of an object-oriented programming language. Mastering object-oriented programming means becoming conversant with the object-oriented concepts and being able to apply them effectively and systematically in developing programs. The book teaches object-oriented programming, and students will learn how to develop object-oriented programs.

The second objective of this book is to prepare students for real-world programming. Knowing object-oriented concepts is not enough. Students must be able to apply that knowledge to develop real-world programs. Sample programs in many introductory textbooks are too simplistic, and they do not teach students techniques to develop large object-oriented programs. In this book, we teach students how to use classes from the class libraries from Chapter 2 and how to define their own classes from Chapter 4. We emphasize foremost the teaching of effective object-oriented design and necessary foundations for building large-scale programs. We will discuss this point further in the Features section of this preface.

New Features in the Third Edition

We would like to take this opportunity to thank the adopters of the earlier editions. Numerous suggestions we received from the adopters and their students helped us tremendously in improving the texbook. For this edition, we focused on improving the strengths of the earlier editions, updating and adding the materials by incorporating capabilities of Java 2 SDK 1.4 class libraries, and removing the materials that have less relevance and significance today. Before we get into the features of the book, we will highlight briefly the changes we made in the third edition:

1. **Full-color pages.** We started with one color in the first edition. We introduced the second color in the second edition to enhance the illustrations and the overall presentation of the materials. We took it one step further and decided to use full-color pages in this edition. The result is dramatic. Different sections are clearly identified, syntax coloring is used in code listings, illustrations are more lucid, and the overall flow of the pages is very attractive and appealing. Pedagogy has been greatly enhanced by the use of full-color pages.

   In addition to the use of full color, the page layout is completely redesigned, with new icons to highlight the helpful reminders and design guidelines. We adopted the Japanese rock garden as our design theme.

2. **No reliance on the javabook classes.** In the earlier versions of Java systems, we did not have an easy way to perform input and output. To work around this shortcoming, many authors provided their own classes which the students can use to perform input and output. To that end, we provided a GUI (graphical user interface) based collection of classes organized into a package named `javabook`. The situation has improved with the introduction of new Swing classes, most notably the `JOptionPane` class. In this edition, we will be using the standard classes exclusively for input and output. The `javabook` package is still available, however, for those who wish to use them. We will discuss more on this in the Features section of this preface.

3. **Short sample programs.** Many reviewers and adopters requested more short sample programs that illustrate the concepts in a succint manner. We now have numerous short sample programs throughout the chapters. At the end of many sections, we provide one or two short sample programs to illustrate the main concept taught in that section. A complete list of all sample classes (programs) is given in Appendix B.

4. **Java 2 SDK 1.4 materials.** Many new features are added to the newest Java SDK version 1.4. Not only our sample programs are compatible

with SDK 1.4, we actively teach newly added features of version 1.4 such as pattern matching capabilities and assertion features.

5. **No applets.** We no longer teach applets in this textbook. After students cover Chapter 7 and a portion of Chapter 14, they can easily master applets on their own. A short handout on applets is available from our website.
6. **Swing classes for GUI.** For teaching GUI and event-driven programming, we use Swing classes exclusively. There will be no discussion on AWT-based GUI components.
7. **UML notations.** UML diagrams are used to document the relationships of the classes in the sample programs. Other diagrams (such as state-of-memory diagrams) will also use UML notations for consistency. Notice that although UML notations are used in these diagrams, these illustrative diagrams are not strictly speaking UML diagrams.

Features

There are many pedagogical features that make this book attractive. We will describe the defining features of this book.

Feature 1

Java

We chose Java for this book. Unlike C++, Java is a pure object-oriented language, and it is an ideal language to teach object-oriented programming because Java is logical and much easier to program when compared to other object-oriented programming languages. Java's simplicity and clean design make it one of the most easy to program object-oriented languages today. Java does not include language features that are too complex and could be a roadblock for beginners in learning object-oriented concepts. Although we use Java, we must emphasize that this book is not about Java programming. It is about object-oriented programming, and as such, we do not cover every aspect of Java. We do, however, cover enough language features of Java to make students competent Java programmers.

Feature 2

Standard Classes for Input and Output

In this edition, we decided to use the standard classes exclusively for both GUI and console input and output. We still make the author-defined javabook classes available for use, but there will be no discussion on their use in the textbook. Also, some exercises may still suggest the use of certain javabook classes, but its use is not mandatory in solving them.

With the advent of Swing classes, specifically the JOptionPane class, the practical reason for using the javabook classes is eliminated for the most part. Moreover, we can achieve most of the pedagogical reasons for providing the javabook classes by using appropriate standard classes such as String, Date, and others instead. We, therefore, decided to drop the javabook classes in this edition.

Feature 3

Full-Immersion Approach

We wrote a series of articles in 1993 on how to teach object-oriented programming in the *Journal of Object-Oriented Programming* (Vol. 6, No. 1; Vol. 6, No. 4; and Vol. 6, No. 5). The core pedagogic concept we described in the series is that one must become an object user before becoming an object designer. In other words, before being able to design one's own classes effectively, one first must learn how to use predefined classes. We adopt a full-immersion approach in which students learn how to use objects from the first program. It is very important to ensure that the core concepts of object-oriented programming are emphasized from the beginning. Our first sample program from Chapter 2 is this:

```
/* Chapter 2 Sample Program: Displays a Window
    File: Ch2Sample1.java
*/

import javax.swing.*;

class Ch2Sample1 {

    public static void main( String[] args ) {

        JFrame myWindow;

        myWindow = new JFrame();

        myWindow.setSize(300, 200);
        myWindow.setTitle("My First Java Program");
        myWindow.setVisible(true);
    }
}
```

This program captures the most fundamental notion of object-oriented programming. That is, an object-oriented program uses objects. As obvious as it may sound, many introductory books do not really emphasize this fact. In the program, we use a `JFrame` object called `myWindow` to display a generic window. Many introductory textbooks begin with a sample program such as

```
/*
   Hello World Program
*/

class HelloWorld
{
    public static void main(String args[])
    {
       System.out.println("Hello World");
    }
 }
```

or

```
/*
    Hello World Applet
*/
import java.applet.*;
import java.awt.*;

public class HelloWorld extends Applet
{
   public void paint( Graphics g)
   {
       g.drawString("Hello World", 50, 50);
   }
}
```

Both programs have problems. They do not illustrate the key concept that object-oriented programs use objects. The first program does indeed use an object System.out, but the use of System.out does not illustrate the object declaration and creation. Beginners normally cannot differentiate classes and objects. So it is very important to emphasize the concept that you need to declare and create an object from a class before you can start using the object. Our first sample program does this.

The second HelloWorld program is an applet, which, as its name suggests, is a mini-application with a very specific usage. Applets are specific to Java, and our objective is to teach object-oriented programming, not to teach the specific features of the Java language.

Feature 4

Illustrations

We believe a picture is worth a thousand words. Difficult concepts can be explained nicely with lucid illustrations. Diagrams are an important tool for designing and documenting programs, and no programmers will develop real-world software applications without using some form of diagramming tools. We use UML diagrams for the sample programs, and UML notations are used consistently in all types of illustrations.

This book includes numerous illustrations that are used as a pedagogic tool to explain core concepts such as inheritance, memory allocation for primitive data types and objects, parameter passing, and others. Representative illustrations can be found on pages 38, 155, 196, 197, 585, 604, 653, 743, and 835.

Feature 5

Incremental Development

We teach object-oriented software engineering principles in this book. Instead of dedicating a separate chapter for the topic, we interweave program development principles and techniques with other topics. Every chapter from Chapter 2 to Chapter 14 includes a sample development to illustrate the topics covered in the chapter, and we develop the program using the same design methodology consistently.

This book teaches a software design methodology that is conducive to object-oriented programming. All sample developments in this book use a technique we characterize as incremental development. The incremental development technique is based on the modern iterative approach (some call it a spiral approach), which is a preferred methodology of object-oriented programmers.

Beginning programmers tend to mix the high-level design and low-level coding details, and their thought process gets all tangled up. Presenting the final program is not enough. If we want to teach students how to develop programs, we must show the development process. An apprentice will not become a master builder just by looking at finished products, whether they are furniture or houses. Software construction is no different.

The problem with other textbooks is that the authors often dedicate a single chapter to discuss and preach effective software development methodologies, but they never actually show how to put these methodologies into practice. They only show and explain the finished products. But without putting what they preach into practice by showing the development process, students will not learn how to develop programs. And it is not enough to show the development process once. We must show the development process repeatedly. In this book, we develop every sample development program incrementally to show students how to develop programs in a logical and methodical manner.

Feature 6

Design Guidelines, Helpful Reminders, and Quick Checks

Throughout the book, we include design guidelines and helpful reminders. Almost every section of the chapters is concluded with a number of Quick Check questions to make sure that students have mastered the basic points of the section.

Design guidelines are indicated with a bonsai icon like this:

Design Guidelines

Design a class that implements a single well-defined task. Do not overburden the class with multiple tasks.

Helpful reminders come in different styles. The first style is indicated with a stone lantern icon like this:

Helpful Reminder

Watch out for the off-by-one error (OBOE).

The second style is a Take My Advice box:

Take my Advice

On occasions, programming can be very frustrating because no amount of effort on your part would make the program run correctly. You are not alone. Professional programmers often have the same feeling, including this humble self. But, if you take time to think through the problem and don't lose your cool, you will find a solution. If you don't, well, it's just a program. Your good health is much more important than a running program and a good grade.

Interesting pieces of background information are presented in the You Might Want to Know box:

You Might Want to Know

There's a reason behind choosing a Japanese rock garden as a design theme. The famous rock garden at Ryoan-ji temple in Kyoto, Japan has 15 stones. It is told that viewing the rock garden from any angle you can see only 14 of these stones, but when you master Zen, you can "see" the 15th stone with your mind's eye. Likewise, when you study object-oriented programming with this book, you can visualize objects easily with your mind's eye while developing programs.

When we refer to materials available on websites that are related to the topic covered in the book, we indicate this availability with the following icon:

www

How-to documents on how to compile and run Java programs with different development tools are available from our website at **www.drcaffeine.com.**

Quick Check questions appear at the end of the sections with the following banner:

Quick CHECK

1. How many stones are there at Ryoan-ji's rock garden?
2. Name the purpose of the bonsai and lantern icons.

Feature 7

Graphical User Interface and Event-Driven Programming

Since modern real-world programs are GUI-based and event-driven, we cannot skirt around them if we want to teach the foundation of real-world programming. Although we teach console input and output and use them in many sample programs, the large sample programs in this book are GUI-based. We introduce Swing-based GUI components in Chapter 7 and present advanced GUI topics in Chapter 14. We feel strongly that GUI and event-driven programming must be taught in CS1, but those instructors who wish to keep the discussion on user interface to a minimum can omit the entire Chapter 14.

Feature 8

Assertions and Pattern Matching

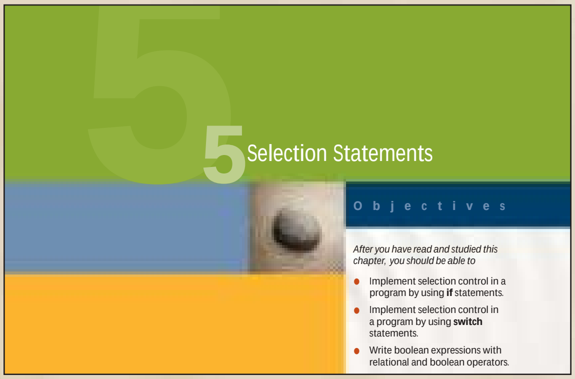
Two of the new features added to Java 2 SDK 1.4 are pattern matching and assertions. Pattern matching is a very powerful and flexible tool in manipulating strings, and we teach pattern matching with many examples in Chapter 9. Assertion is one of the software engineering techniques to ensure the program reliability, and finally, the assertion feature is added to Java. In Chapter 8, we explain two key language features—exception handling and assertion—which we can use to improve program reliability.

Walk Through

This guided tour is designed to walk you through the features of the chapters and the supplements. As you examine them, note the following

- Each chapter begins by orienting you to what you will learn.
- A variety of examples are used to demonstrate concepts.
- A large number of colorful diagrams intuitively explains concepts.
- Excellent pedagogy keeps students motivated.
- Each chapter reinforces concepts in numerous ways.

Chapter openers orient students to what they will learn



5 Selection Statements

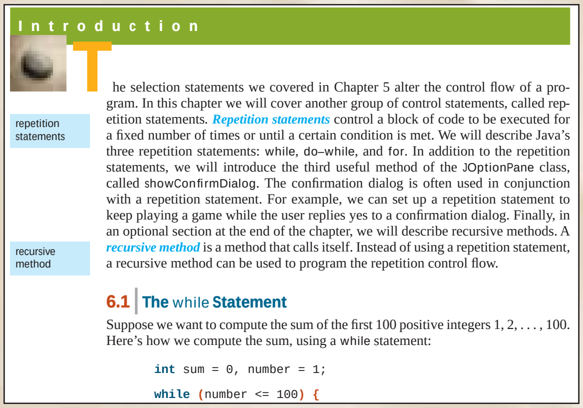
Objectives

After you have read and studied this chapter, you should be able to

- Implement selection control in a program by using **if** statements.
- Implement selection control in a program by using **switch** statements.
- Write boolean expressions with relational and boolean operators.

Chapter Objectives detail what students will be able to accomplish when they have worked through the chapter.

The Introduction motivates the material to be covered in the chapter and often relates the topics to be learned to what has already been covered.



Introduction

repetition statements

recursive method

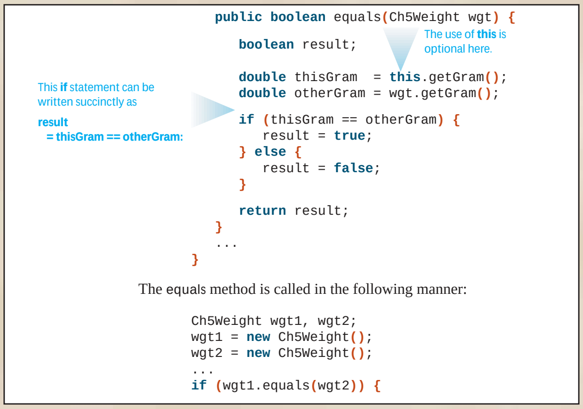
The selection statements we covered in Chapter 5 alter the control flow of a program. In this chapter we will cover another group of control statements, called repetition statements. ***Repetition statements*** control a block of code to be executed for a fixed number of times or until a certain condition is met. We will describe Java's three repetition statements: while, do–while, and for. In addition to the repetition statements, we will introduce the third useful method of the JOptionPane class, called showConfirmDialog. The confirmation dialog is often used in conjunction with a repetition statement. For example, we can set up a repetition statement to keep playing a game while the user replies yes to a confirmation dialog. Finally, in an optional section at the end of the chapter, we will describe recursive methods. A ***recursive method*** is a method that calls itself. Instead of using a repetition statement, a recursive method can be used to program the repetition control flow.

6.1 The while Statement

Suppose we want to compute the sum of the first 100 positive integers 1, 2, . . . , 100. Here's how we compute the sum, using a while statement:

```
int sum = 0, number = 1;

while (number <= 100) {
```

Teaching By Example



```
        public boolean equals(Ch5Weight wgt) {

           boolean result;

           double thisGram  = this.getGram();
           double otherGram = wgt.getGram();

           if (thisGram == otherGram) {
              result = true;
           } else {
              result = false;
           }

           return result;
        }
        ...
     }
```

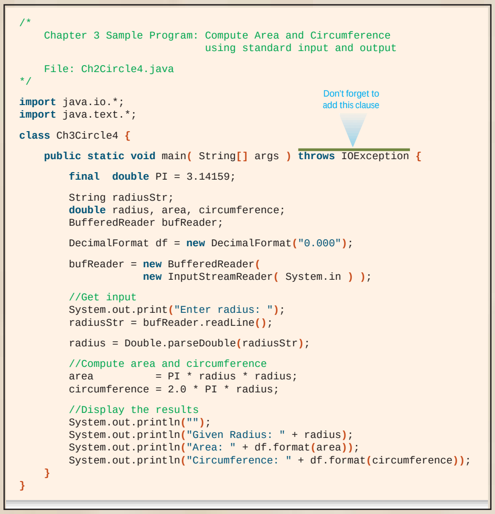
The use of this is optional here.
This if statement can be written succinctly as
result
= thisGram == otherGram;

The equals method is called in the following manner:

```
Ch5Weight wgt1, wgt2;
wgt1 = new Ch5Weight();
wgt2 = new Ch5Weight();
...
if (wgt1.equals(wgt2)) {
```

Code with comments is found throughout the text. This annotated code helps students to understand how the various lines of code work.

Short Example Programs scattered throughout each chapter demonstrate how to implement concepts being learned.



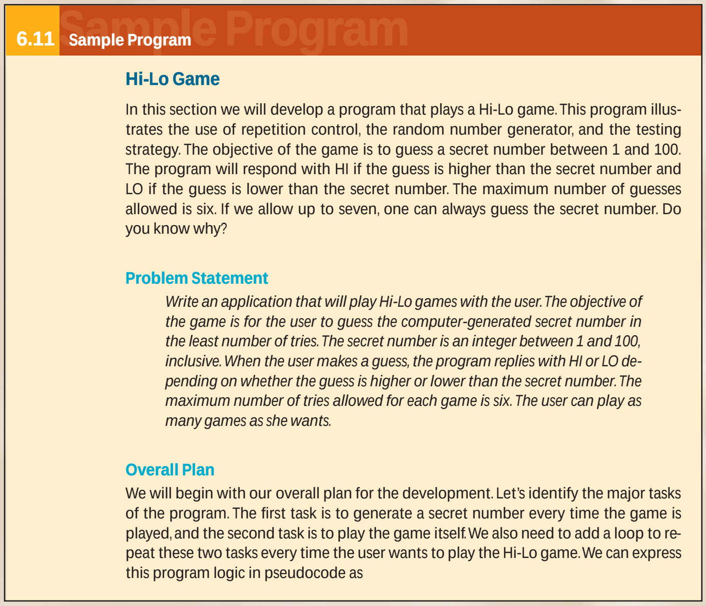
```
/*
    Chapter 3 Sample Program: Compute Area and Circumference
                              using standard input and output

    File: Ch2Circle4.java
*/

import java.io.*;
import java.text.*;

class Ch3Circle4 {

    public static void main( String[] args ) throws IOException {

        final  double PI = 3.14159;

        String radiusStr;
        double radius, area, circumference;
        BufferedReader bufReader;

        DecimalFormat df = new DecimalFormat("0.000");

        bufReader = new BufferedReader(
                    new InputStreamReader( System.in ) );

        //Get input
        System.out.print("Enter radius: ");
        radiusStr = bufReader.readLine();

        radius = Double.parseDouble(radiusStr);

        //Compute area and circumference
        area          = PI * radius * radius;
        circumference = 2.0 * PI * radius;

        //Display the results
        System.out.println("");
        System.out.println("Given Radius: " + radius);
        System.out.println("Area: " + df.format(area));
        System.out.println("Circumference: " + df.format(circumference));
    }
}
```

Longer Sample Programs at the end of each chapter walk students through larger examples, giving students a framework for building programs incrementally.

Problem Statement—presents the goal of the program to be designed.

Overall Plan—at this stage the problem is broken down into tasks and a plan is developed.

Development: Then for each component to the program that needs to be developed the student is walked through the steps of design, coding and testing, then finalizing the program.



6.11 Sample Program

Hi-Lo Game

In this section we will develop a program that plays a Hi-Lo game. This program illustrates the use of repetition control, the random number generator, and the testing strategy. The objective of the game is to guess a secret number between 1 and 100. The program will respond with HI if the guess is higher than the secret number and LO if the guess is lower than the secret number. The maximum number of guesses allowed is six. If we allow up to seven, one can always guess the secret number. Do you know why?

Problem Statement

Write an application that will play Hi-Lo games with the user. The objective of the game is for the user to guess the computer-generated secret number in the least number of tries. The secret number is an integer between 1 and 100, inclusive. When the user makes a guess, the program replies with HI or LO depending on whether the guess is higher or lower than the secret number. The maximum number of tries allowed for each game is six. The user can play as many games as she wants.

Overall Plan

We will begin with our overall plan for the development. Let's identify the major tasks of the program. The first task is to generate a secret number every time the game is played, and the second task is to play the game itself. We also need to add a loop to repeat these two tasks every time the user wants to play the Hi-Lo game. We can express this program logic in pseudocode as

Visual Approach

Diagrams give visual representation to concepts and help to explain the relationships of classes and objects.

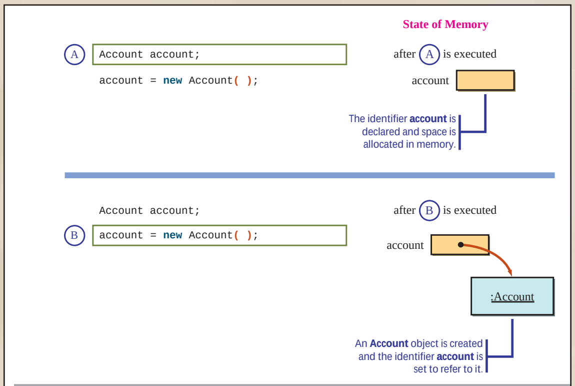

State of Memory
A
Account account;
account = new Account( );
after A is executed
account
The identifier account is declared and space is allocated in memory.
Account account;
B
account = new Account( );
after B is executed
account
:Account
An Account object is created and the identifier account is set to refer to it.

Figure 2.4 Distinction between object declaration and object creation.

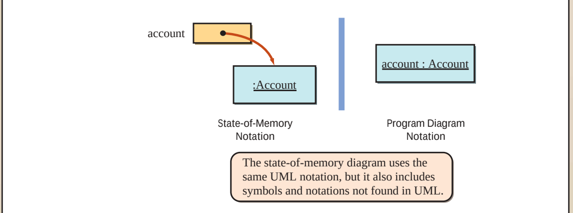
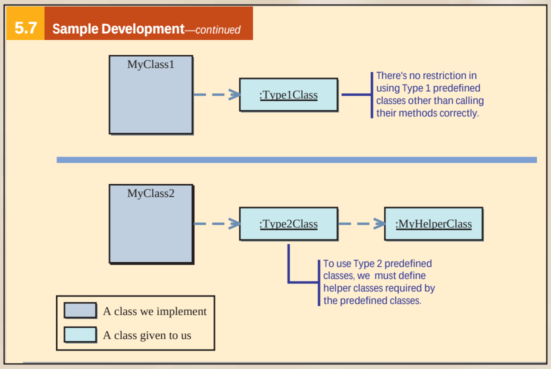
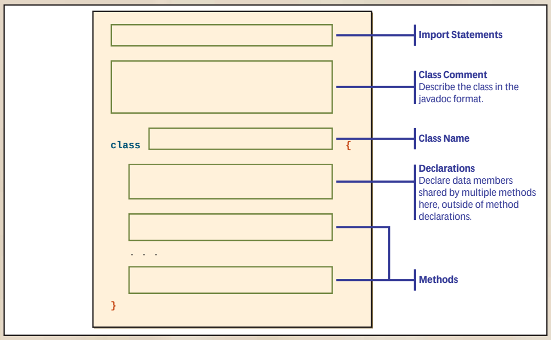

account
:Account
account : Account
State-of-Memory Notation
Program Diagram Notation
The state-of-memory diagram uses the same UML notation, but it also includes symbols and notations not found in UML.

Figure 2.5 Relationship between the state-of-memory diagram and the program diagram notation.


5.7 Sample Development—continued
MyClass1
:Type1Class
There's no restriction in using Type 1 predefined classes other than calling their methods correctly.
MyClass2
:Type2Class
:MyHelperClass
To use Type 2 predefined classes, we must define helper classes required by the predefined classes.
A class we implement
A class given to us


class
{
. . .
}
Import Statements
Class Comment
Describe the class in the javadoc format.
Class Name
Declarations
Declare data members shared by multiple methods here, outside of method declarations.
Methods

Excellent Pedagogy

Helpful Reminders provide tips for students to help them become more effective programmers.

Helpful Reminder

To draw geometric shapes on the content pane of a frame window, remember that

1. *The content pane is declared as a* **Container,** *for example,*

```
Container contentPane;
```

2. *The frame window must be visible on the screen before we can get the content pane's* **Graphics** *object.*

You Might Want to Know

To show you just how common the off-by-one error occurs in everyday life, consider the following two questions. When you want to put a fence post every 10 ft, how many posts do you need for a 100-ft fence? If it takes 0.5 seconds for an elevator to rise one floor, how long does it take to reach the fourth floor from the ground level? The answers that come immediately are 10 posts and 2 seconds, respectively. But after a little more thought, we realize the correct answers are 11 posts (we need the final post at the end) and 1.5 seconds (there are three floors to rise to reach the fourth floor from the ground level).

You Might Want to Know boxes give students interesting bits of information

Take My Advice boxes give students advice on coding from an experienced programmer perspective.

Take my Advice

It takes some practice before you can write well-formed **if** statements. Here are some rules to help you write the **if** statements.

Rule 1:	Minimize the number of nestings.
Rule 2:	Avoid complex boolean expressions. Make them as simple as possible. Don't include many ANDs and ORs.
Rule 3:	Eliminate any unnecessary comparisons.
Rule 4:	Don't be satisfied with the first correct statement. Always look for improvement.
Rule 5:	Read your code again. Can you follow the statement easily? If not, try to improve it.

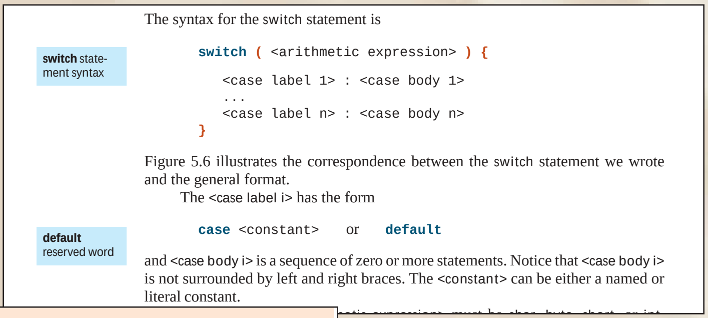
Design Guidelines

Always define a constructor and initialize instance variables fully in the constructor so an object will be created in a valid state.

Design Guidelines provide tips on good program design techniques.

Tools That Reinforce the Concepts

Key Terms are highlighted in the margin, so students can find them easily when studying.


The syntax for the switch statement is

switch statement syntax

```
switch ( <arithmetic expression> ) {
   <case label 1> : <case body 1>
   ...
   <case label n> : <case body n>
}
```

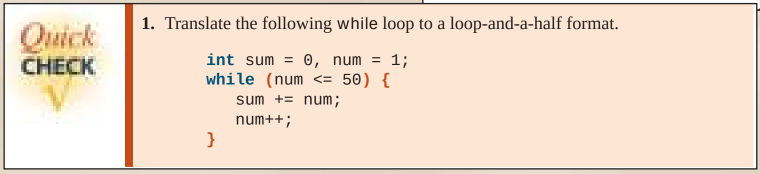
Figure 5.6 illustrates the correspondence between the switch statement we wrote and the general format.

The <case label i> has the form

default reserved word

```
case <constant>    or    default
```

and <case body i> is a sequence of zero or more statements. Notice that <case body i> is not surrounded by left and right braces. The <constant> can be either a named or literal constant.


Quick CHECK

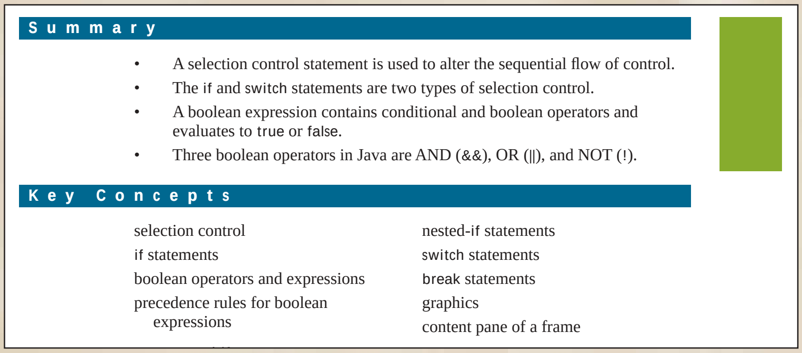
1. Translate the following while loop to a loop-and-a-half format.

```
int sum = 0, num = 1;
while (num <= 50) {
   sum += num;
   num++;
}
```

Quick Check exercises at the end of sections allow students to test their comprehension of concepts.

End of Chapter Summary provides a bulleted explanation of important topics explained in the chapter.

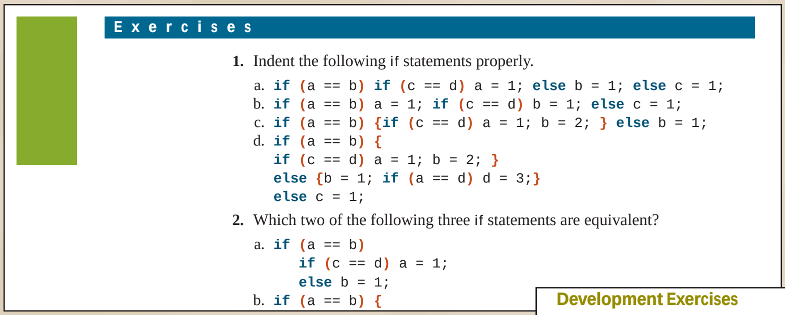
The **Key Concepts** section lists the important terms that the students should know after finishing the chapter.


Summary

- A selection control statement is used to alter the sequential flow of control.
- The if and switch statements are two types of selection control.
- A boolean expression contains conditional and boolean operators and evaluates to true or false.
- Three boolean operators in Java are AND (&&), OR (||), and NOT (!).

Key Concepts

selection control	nested-if statements
if statements	switch statements
boolean operators and expressions	break statements
precedence rules for boolean expressions	graphics
	content pane of a frame


Exercises

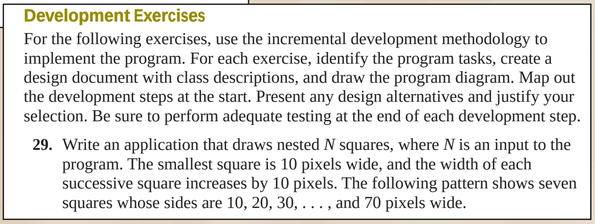
1. Indent the following if statements properly.

```
a. if (a == b) if (c == d) a = 1; else b = 1; else c = 1;
b. if (a == b) a = 1; if (c == d) b = 1; else c = 1;
c. if (a == b) {if (c == d) a = 1; b = 2; } else b = 1;
d. if (a == b) {
   if (c == d) a = 1; b = 2; }
   else {b = 1; if (a == d) d = 3;}
   else c = 1;
```

2. Which two of the following three if statements are equivalent?

```
a. if (a == b)
      if (c == d) a = 1;
      else b = 1;
b. if (a == b) {
```

Exercises at the end of each chapter give students a chance to practice what they are learning.


Development Exercises

For the following exercises, use the incremental development methodology to implement the program. For each exercise, identify the program tasks, create a design document with class descriptions, and draw the program diagram. Map out the development steps at the start. Present any design alternatives and justify your selection. Be sure to perform adequate testing at the end of each development step.

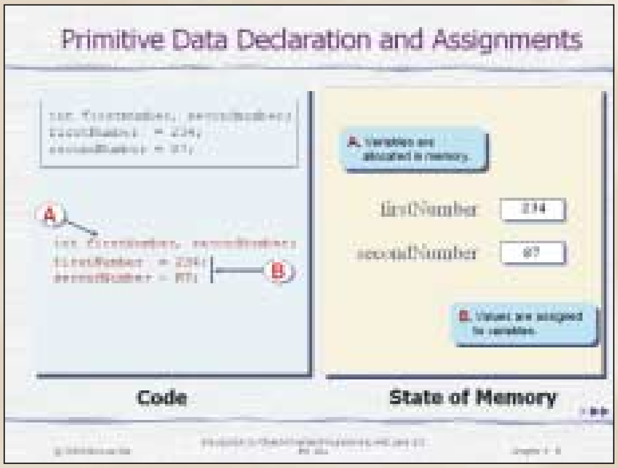
29. Write an application that draws nested *N* squares, where *N* is an input to the program. The smallest square is 10 pixels wide, and the width of each successive square increases by 10 pixels. The following pattern shows seven squares whose sides are 10, 20, 30, . . . , and 70 pixels wide.

Development Exercises give students an opportunity to use the incremental development methodology to implement programs.

Supplements for Instructors and Students

For Instructors

- Complete set of **PowerPoints,** including lecture notes and figures.



Primitive Data Declaration and Assignments

```
int firstNumber, secondNumber;
firstNumber  = 234;
secondNumber = 87;
```

A
B
A. Variables are allocated in memory.
firstNumber 234
secondNumber 87
B. Values are assigned to variables.
Code
State of Memory

- **Complete solutions** for the exercises
- **Example Bank**—Additional examples, which are searchable by topic, are provided online in a “bank” for instructors.
- **Homework Manager/Test Bank**—Conceptual review questions are stored in this electronic question bank and can be assigned as exam questions or homework.
- **Online labs** which accompany this text, can be used in a closed lab, open lab, or for assigned programming projects.

For Students

- **Compiler How Tos** provide tutorials on how to get up and running on the most popular compilers to aid students in using IDEs.

4.2 Creating a Project

- **Source code** for all example programs in the book.
- **Answers** to quick check exercises.
- **Glossary** of key terms.
- **Recent News** links relevant to computer science.
- **Additional Topics** such as more on swing and an introduction to data structures.

Book Organization

There are 16 chapters in this book, numbered from 0 to 15. There are more than enough topics for one semester. Basically the chapters should be covered in linear sequence, but nonlinear sequence is possible. We show the dependency relationships among the chapters at the end of this section.

Here is a short description for each chapter:

- **Chapter 0** is an optional chapter. We provide background information on computers and programming languages. This chapter can be skipped or assigned as an outside reading if you wish to start with object-oriented programming concepts.
- **Chapter 1** provides a conceptual foundation of object-oriented programming. We describe the key components of object-oriented programming and illustrate each concept with a diagrammatic notation using UML.
- **Chapter 2** covers the basics of Java programming and the process of editing, compiling, and running a program. From the first sample program presented in this chapter, we emphasize object-orientation. We will introduce the standard classes String, JOptionPane, Date, and SimpleDateFormat so we can reinforce the notion of object declaration, creation, and usage. Moreover, by using these standard classes, students can immediately start writing practical programs.
- **Chapter 3** introduces variables, constants, and expressions for manipulating numerical data. We explain the standard Math class from java.lang and introduce more standard classes (GregorianCalendar and DecimalFormat) to continually reinforce the notion of object-orientation. We describe and illustrate console input and output (System.in and System.out) so the students can see the use of non-GUI I/O. The optional section explains how the numerical values are represented in memory space.
- **Chapter 4** teaches how to define and use your own classes. The key topics covered in this chapter are constructors, visibility modifiers (public and private), local variables, parameter passing, and value-returning methods. We explain and illustrate parameter passing and value-returning methods using both primitive data types (int, double, etc.) and reference data types (objects). Through these explanations and illustrations, we clearly distinguish the primitive and reference data types. By the end of this chapter, students will have a solid understanding of object-orientation.
- **Chapter 5** explains the selection statements if and switch. We cover boolean expressions and nested-if statements. We explain how objects are compared

by using equivalence (==) and equality (the equals and compareTo methods). Illustrative and meaningful examples are provided to make the distinction between the equivalence and equality clear. Drawing 2-D graphics is introduced, and a screensaver sample development program is developed.

- **Chapter 6** explains the repetition statements while, do–while, and for. Pitfalls in writing repetition statements are explained. The use of confirmation dialog with the showConfirmDialog method of JOptionPane is shown. The optional last section of the chapter introduces recursion as another technique for repetition.
- **Chapter 7** covers basic GUI components and event-driven programming. Only the Swing-based GUI components are covered in this chapter. This chapter provides a first glimpse of using inheritance. We limit the discussion to defining a subclass of a standard class (JFrame) as a nice foundation for a fuller coverage of inheritance in Chapter 13. GUI components introduced in this chapter are JButton, JLabel, ImageIcon, JTextField, JTextArea, and menu-related classes. Our main focus for this chapter is event-driven programming, so we keep the discussion on GUI components at the basic level. For instance, we defer the discussion on layout managers and mouse events until Chapter 14. For those who wish to cover more GUI topics can teach a portion of Chapter 14 before continuing to Chapter 8.
- **Chapter 8** teaches exception handling and assertions. The focus of this chapter is the construction of reliable programs. We provide a detailed coverage of exception handling in this chapter. In the previous edition, we presented exception handling as a part of discussing file input and output. In this edition, we treat it as a separate topic. We introduce an assertion, a newly added Java 2 SDK 1.4 feature, and show how it can be used to improve the reliability of finished products by catching logical errors early in the development.
- **Chapter 9** covers nonnumerical data types: characters and strings. Both the String and StringBuffer classes are explained in the chapter. An important application of string processing is pattern matching. We describe pattern matching and regular expression in this chapter. We introduce the Pattern and Matcher classes, newly added to Java 2 SDK 1.4 and show how they are used in pattern matching.
- **Chapter 10** teaches arrays. We cover arrays of primitive data types and of objects. An array is a reference data type in Java, and we show how arrays are passed to methods. We describe how to process two-dimensional arrays and explain that a two-dimensional array is really an array of arrays in Java. Lists and maps are introduced as more general and flexible ways to maintain a collection of data. The use of ArrayList and HashMap classes from the java.util package is shown in the sample programs. Also, we show how the WordList helper class used in Chapter 9 sample development program is implemented with another map class called TreeMap.
- **Chapter 11** presents searching and sorting algorithms. Both N^2 and $N \log_2 N$ sorting algorithms are covered. The mathematical analysis of searching and sorting algorithms can be omitted depending on the students' background.

- **Chapter 12** explains the file I/O. Standard classes such as `File` and `JFileChooser` are explained. We cover all types of file I/O, from a low-level byte I/O to a high-level object I/O. We show how the file I/O techniques are used to implement the helper classes—`Dorm` and `FileManager`—in Chapter 8 and 9 sample development programs.
- **Chapter 13** discusses inheritance and polymorphism and how to use them effectively in program design. The effect of inheritance for member accessibility and constructors is explained. We also explain the purpose of abstract classes and abstract methods.
- **Chapter 14** covers advanced GUI. We describe the effective use of nested panels and layout managers. Handling of mouse events is described and illustrated in the sample programs. A capstone sample development program that uses a modified Model-View-Controller design pattern is constructed in this chapter.
- **Chapter 15** covers recursion. Because we want to show the examples where the use of recursion really shines, we did not include any recursive algorithm (other than those used for explanation purposes) that really should be written nonrecursively.

Chapter Dependency

For the most part, chapters must be read in sequence, but some variations are possible, especially with the optional chapters. Chapters 0, 14, 15 and Section 6.12 are optional. Section 8.6 on assertions can be considered optional. Here's a simplified dependency graph (optional chapters are shown in green):

$$\begin{tikzcd}
0 \arrow[r] & 1 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & 4 \arrow[r] & 5 \arrow[r] & 6 \arrow[r] \arrow[dl] \arrow[dr] & 7 \arrow[r] & 14 \\
& & & & & 8 \arrow[dl] \arrow[d] \arrow[dr] & & 10 \arrow[dl] \arrow[d] \arrow[dr] & \\
& & & & 9 & 12 & 13 & 11 & 15
\end{tikzcd}$$

More detailed information on chapter dependency and suggested sequences for different audiences can be found at our website.

Acknowledgments

I would like to thank the following reviewers and the focus group participants for their comments, suggestions, and encouragement.

Focus Group Attendees

Roger Ferguson, *Grand Valley State University*
Gerald Gordon, *Depaul University*
Susan Haller, *University of Wisconsin, Parkside*
Eliot Jacobson, *University of California, Santa Barbara*
Marian Manyo, *Marquette University*
Thaddeus Pawlicki, *University of Rochester*
Paul Tymann, *Rochester Institute of Technology*

Reviewers:

Ken Brown, *University of Aberdeen, UK*
Robert Burton, *Brigham Young University*
Michael Crowley, *University of Southern California*
Adrienne Decker, *University of Buffalo*
Deborah Deppeler, *University of Wisconsin, Madison*
Julian Dermoudy, *University of Tasmania, Australia*
Roger Ferguson, *Grand Valley State University*
Mark Fienup, *University of Northern Iowa*
H.J. Geers, *Technical University Delft, Netherlands*
John Hamer, *University of Auckland, New Zealand*
Sherri Harms, *University of Nebraska, Lincoln*
Joseph D. Hurley, *Texas A & M University*
Eliot Jacobson, *University of California, Santa Barbara*
Saroja Kanchi, *Kettering University*
Andrew Kinley, *Rose-Hulman Institute of Technology*
Blayne E. Mayfield, *Oklahoma State University*
James McElroy, *California State University, Chico*
Carolyn S. Miller, *North Carolina State University*
Jayne Valenti Miller, *Purdue University*
Thaddeus Pawlicki, *University of Rochester*
Gyorgy Petruska, *Indiana University-Purdue University, Fort Wayne*
David Raymond, *United States Military Academy, West Point*
Donna S. Reese, *Mississippi State University*
Alan Saleski, *Loyola University*
Carolyn Schauble, *Colorado State University*
Ken Slonneger, *University of Iowa*
Howard Straubing, *Boston College*
Alex Thornton, *University of California, Irvine*
David Vineyard, *Kettering University*
Gregory F. Welch, *University of North Carolina*

I would like to thank the following McGraw-Hill staff for their trust in my ability to produce a quality text book and their never-ending support throughout the whole project.

Kelly Lowery
Emily Lupash
Sheila Frank
Dawn Bercier

My Story

In September, 2001, I changed my name for personal reasons. Prof. Thomas Wu is now Prof. Thomas Otani. To maintain continuity and not to confuse people, the third edition is published under my former name. Those who care to find out the reasons for changing the name (they are not dramatic) can do so by visiting my website (www.drcaffeine.com).