

# Preface

**A**n *Introduction to Object-Oriented Programming with Java* takes a full-immersion approach to object-oriented programming. Proper object-oriented design practices are emphasized throughout the book. Students learn to be object users first, then learn to be class designers. In the fourth edition, we use a gentler approach to teaching students how to design their own classes, separating the coverage into two chapters.

## Key Changes in the Fourth Edition

Before we get into the features of the book, first we will highlight briefly the changes we made in the fourth edition. The fourth edition includes more accessible and in-depth discussion of programmer-defined classes, Java 5.0 (Java 2 SDK 1.5) topics, and less dependency on GUI.

- 1. Gentler Introduction to Programmer-Defined Classes.** One of the most difficult aspects for students in learning object-oriented programming is the creation of programmer-defined classes. Most students find using objects from the standard classes fairly straightforward. However, they frequently stumbled when trying to define their own classes. In the third edition, we presented all topics related to programmer-defined classes in one chapter. In the fourth edition, we spread the topics to two chapters. We present the basics of programmer-defined classes in Chapter 4 with new examples and gentler discussions more accessible to students.
- 2. More In-depth Coverage of Programmer-Defined Classes.** In Chapter 7, we present more in-depth coverage of programmer-defined classes, including topics, such as, method overloading, the use of the reserved word `this`, and class methods and variables. These topics are the ones most students find difficult to grasp. By deferring the advanced topics until Chapter 7, after the traditional topics on selection and repetition controls are covered, students are more prepared to understand them. Also, by using control structures, we can

present these OO features with more detailed and realistic examples that clearly show the needs for such features.

- 3. Java 5.0 (also known as Java 2 SDK 1.5).** The latest Java 2 SDK includes many additions. In the fourth edition, we describe some of them that improve the teaching of CS1. The first is the `Scanner` class. Prior to SDK 1.5, standard input routines are done by using a `BufferedReader` object. Since a `BufferedReader` throws an exception, we must either discuss exception handling before teaching the standard input or provide some kind of author-defined input class that hides the exception handling. With the new `Scanner` class we can teach much simpler input routines that do not require any exception handling. We introduce the `Scanner` class in Chapter 3. The second is the `Formatter` class. This class provides the formatting technique almost identical to the one supported by the C programming language. We teach the `Formatter` class in Chapter 6.
- 4. No Dependency on GUI.** In the third edition, we introduced basic GUI and event-driven programming in Chapter 7 and advanced GUI in Chapter 14. Some of the examples and sample developments in later chapters require the knowledge of GUI. We combined them into one chapter and moved the combined chapter to Chapter 14, thus providing flexibility. Those instructors who do not teach GUI in the CS1 course can use the fourth edition as is. Those who teach GUI can choose to cover selected GUI topics and introduce them as early as after Chapter 2.

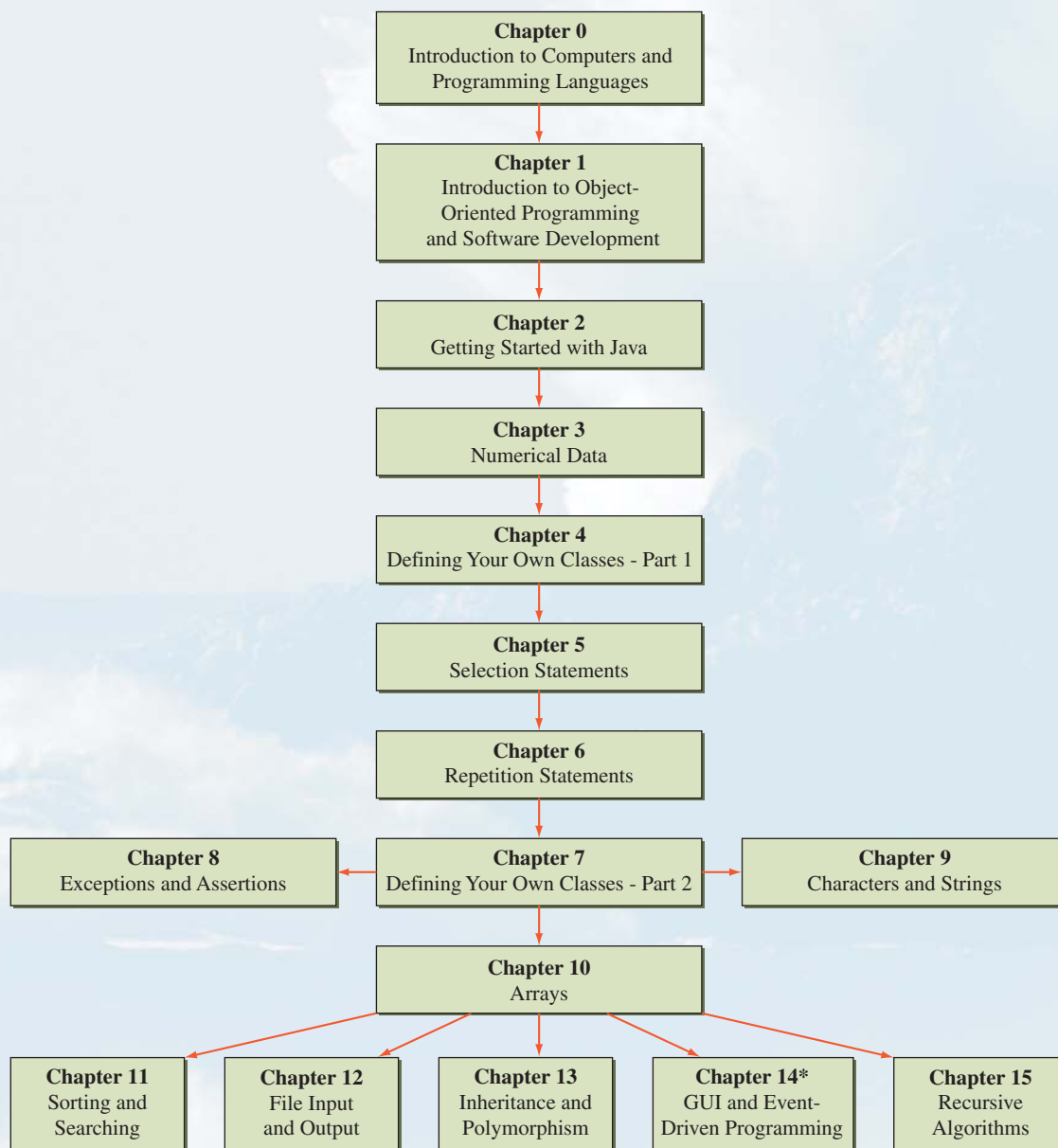
### Book Organization

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 first show the dependency relationships among the chapters and then provide a brief summary of each chapter

### Chapter Dependency

For the most part, chapters should be read in sequence, but some variations are possible, especially with the optional chapters. Chapters 0, 14, 15 and Sections 3.

and 6.12 are optional. Section 8.6 on assertions can be considered optional. Here's a simplified dependency graph:



\*Note: Some examples use arrays, but the use of arrays is not an integral part of the examples. These examples can be modified to those that do not use arrays. Many topics from the early part of the chapter can be introduced as early as after Chapter 2.

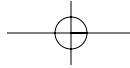
## Brief Chapter Summary

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 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 with `System.in` and the new `Scanner` class and output with `System.out`. The optional section explains how the numerical values are represented in memory space.
- **Chapter 4** teaches the basics of creating programmer-defined classes. We keep the chapter accessible by introducing only the fundamentals with illustrative examples. The key topics covered in this chapter are constructors, visibility modifiers (`public` and `private`), local variables, and passing data to methods. We provide easy-to-grasp illustrations that capture the essence of the topics so the students will have a clear understanding of them.
- **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). We use the `String` and the programmer-defined `Fraction` classes to make the distinction between the equivalence and equality clear. Drawing 2-D graphics is introduced, and a screen saver sample development program is developed.
- **Chapter 6** explains the repetition statements `while`, `do-while`, and `for`. Pitfalls in writing repetition statements are explained. One of the pitfalls to avoid is the use of `float` or `double` for the data type of a counter variable. We illustrate this pitfall by showing a code that will result in an infinite loop. Finding the greatest common divisor of two integers is used as an example of a nontrivial loop statement. We show the difference between the straightforward (brute-force) and the clever (Euclid's) solutions. The use of confirmation dialog with the `showConfirmDialog` method of `JOptionPane` is shown. We introduce the `Formatter` class (new to Java 2 SDK 1.5) and show how the output can be aligned nicely. The optional last section of the chapter introduces recursion as

another technique for repetition. The recursive version of a method that finds the greatest common divisor of two integers is given.

- **Chapter 7** is the second part of creating programmer-defined classes. We introduce new topics related to the creation of programmer-defined classes and also repeat some of the topics covered in Chapter 4 in more depth. The key topics covered in this chapter are method overloading, the reserved word `this`, class methods and variables, returning an object from a method, and pass-by-value parameter passing. As in Chapter 4, we provide many lucid illustrations to make these topics accessible to beginners. We use the `Fraction` class to illustrate many of these topics, such as the use of `this` and class methods. The complete definition of the `Fraction` class is presented in this chapter.
- **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 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 and show how they are used in pattern matching.
- **Chapter 10** teaches arrays. We cover arrays of primitive data types and 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 a more general and flexible way 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  $M\log_2 N$  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 GUI and event-driven programming. Only the Swing-based GUI components are covered in this chapter. GUI components introduced in this chapter include JButton, JLabel, ImageIcon, JTextField, JTextArea, and menu-related classes. We describe the effective use of nested panels and layout managers. Handling of mouse events is described and illustrated in the sample programs. Those who do not teach GUI can skip this chapter altogether. Those who teach GUI can introduce the beginning part of the chapter as early as after Chapter 2.
- **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.

## Hallmark Features of the Text

### Problem Solving

8.7 Sample Development

**Keyless Entry System**

We will develop a program that simulates a secure keyless entry system for a dormitory. Inside the entrance hall of a dorm, there is an entry system where the dorm residents must enter their names, room numbers, and passwords. Upon entry of valid data, the system will unlock the inner door that leads to the dorm's living quarters. To implement this program, two helper classes are provided. The **Door** class simulates unlocking of the inner door. The **Dorm** class manages resident information. An instance of the **Dorm** class is capable of adding and deleting resident information, reading and saving resident information from and to a file, and retrieving information if given the resident's name. We can verify the validity of the entered data by checking them against the information kept by a **Dorm** object.

You  
Might  
Want to  
Know

We can turn our simulation program into a real one by replacing the **Door** class with a class that actually controls the door. Java provides a mechanism called Java Native Interface (JNI) which can be used to embed a link to a low-level device driver code, so calling the **open** method actually unlocks the door.

**Problem Statement**

*Implement a sentry program that asks for three pieces of information: resident's name, room number, and a password. A password is any sequence of characters ranging in length from 4 to 8 and is unique to an individual dorm resident. If everything matches, then the system unlocks and opens the door. We assume no two residents have the same name. Use the provided support classes **Door** and **Dorm**.*

**Overall Plan**

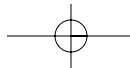
To provide a complete system, we actually have to write two separate programs. The first one is the administrative module for adding, removing, and updating the resident information. The second is the user module that interacts with the residents. Figure 8.8 shows the program diagrams for the two modules.

In this section, we implement the user module. The administrative module is left as an exercise. To begin our development effort, we must first find out the capabilities of the **Dorm** and **Door** classes. Also, for us to implement the class correctly, we need the specification of the **Resident** class.

### Sample Development Programs

In Chapter 2 to Chapter 13 demonstrate the following important problem solving steps:

- Problem Statement
- Overall Plan
- Design
- Code
- Test



**Development Exercises** gives students an opportunity to practice incremental design.

#### 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.

8. In the sample development, we developed the user module of the keyless entry system. For this exercise, implement the administrative module that allows the system administrator to add and delete `Resident` objects and modify information on existing `Resident` objects. The module will also allow the user to open a list from a file and save the list to a file. Is it proper to implement the administrative module by using one class? Wouldn't it be a better design if we used multiple classes with each class doing a single, well-defined task?
9. Write an application that maintains the membership lists of five social clubs in a dormitory. The five social clubs are the Computer Science Club, Biology Club, Billiard Club, No Sleep Club, and Wine Tasting Club. Use the `Dorm` class to manage the membership lists. Members of the social clubs are

#### Differentiating Assertions and Exceptions

Because both the assertion and the exception mechanisms are intended to improve the program reliability, their use is often mixed up. For example, if we are not attentive, we could end up using the assertion feature wrongly in places where exception-handling routines should be used. Consider the following case. In defining the deposit and the withdraw methods, we did not bother to check the value of the parameter (for the sake of a simplified class definition). The passed amount must be greater than zero for the methods to work correctly. How shall we include such testing? One possibility (a wrong approach) is to use the assertion feature as (we only show the withdraw method).

```
public void withdraw(double amount) {
    assert amount > 0;
    double oldBalance = balance;
    balance -= amount;
    assert balance < oldBalance;
}
```

The “Bad Version” of code feature shows students not only common mistakes but also bad design decisions. This is followed up with the proper solution, giving students an understanding of why their first attempts are incorrect.

## Object-oriented Approach

Wu's full-immersion approach to object oriented programming emphasizes proper object-oriented design practices and use of the Java language from the start. Thus, not only are students told how to be object-oriented programmers, but they are shown by example throughout the text.

```
/*
Chapter 2 Sample Program: Displaying 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);
    }
}
```

Diagrams are used extensively to illustrate key concepts

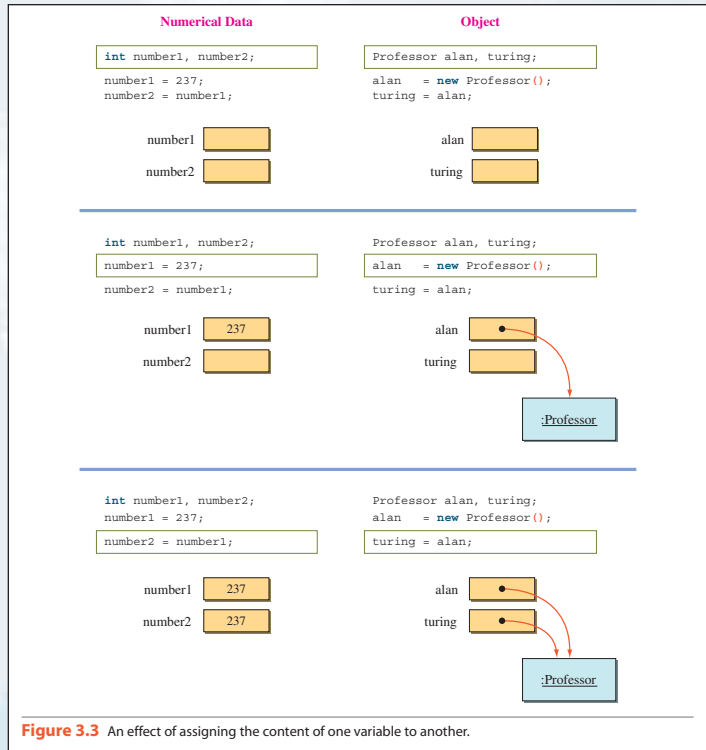


Figure 3.3 An effect of assigning the content of one variable to another.

```

public class Fraction {
    /** the numerator of this fraction */
    private int numerator;
    /** the denominator of this fraction */
    private int denominator;
    //-----
    // Constructors
    //-----
    /**
     * Creates a fraction 0/1
     */
    public Fraction() {
        this(0, 1);
    }
    /**
     * Creates a fraction number/1
     * @param number the numerator
     */
    public Fraction(int number) {
        this(number, 1);
    }
    /**
     * Creates a copy of frac
     * @param frac a copy of this parameter is created
     */
    public Fraction(Fraction frac) {
        this(frac.getNumerator(), frac.getDenominator());
    }
    /**
     * Creates a fraction num/denom

```

Data Members

Constructors

Example code are all written in accordance with good object-oriented practices.

Javadoc comments are used from Chapter 7 sample code to teach students the standard documentation technique for Java code.

```

/**
 * Returns the quotient of this Fraction
 * divided by the parameter frac. The quotient
 * returned is NOT simplified.
 *
 * @param frac the divisor of the division
 *
 * @return the quotient of this fraction
 *         divided by frac
 */
public Fraction divide(Fraction frac) {
    int a, b, c, d;

    Fraction quotient;

    a = this.getNumerator();
    b = this.getDenominator();
    c = frac.getNumerator();
    d = frac.getDenominator();

    quotient = new Fraction(a*d, b*c);

    return quotient;
}

/**
 * Returns the quotient of this Fraction
 * divided by the int parameter number. The quotient
 * returned is NOT simplified.
 *
 * @param number the divisor
 *
 * @return the quotient of this Fraction divided by number
 */
public Fraction divide(int number) {
    Fraction frac = new Fraction(number, 1);
    Fraction quotient = divide(frac);
    return quotient;
}

/**
 * Compares this fraction and the parameter frac for
 * equality. This method compares the two by first

```

## Input/Output

The text uses the latest standard java libraries for user interfaces. Students are introduced to both console-based and graphical approaches with multiple examples of each.

```

/*
 * Chapter 2 Sample Program: Reads a String Input
 * File: Ch2Greetings.java
 */
import javax.swing.*;

class Ch2Greetings {
    public static void main( String[] args ) {
        String name;
        name = JOptionPane.showInputDialog(null, "What is your name?");
        JOptionPane.showMessageDialog(null, "Nice to meet you, "
            + name + ".");
    }
}

```

JOptionPane is used for both input and output in many example programs.

System.out is used to teach console output.

```
final double PI = 3.14159;
String radiusStr;
double radius, area, circumference;

DecimalFormat df = new DecimalFormat("0.000");

//Get input
radiusStr = JOptionPane.showInputDialog(null, "Enter radius:");
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));
```

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

File: Ch2Circle4.java
*/
import java.util.*;
import java.text.*;

class Ch3Circle4 {
    public static void main( String[] args ) {
        final double PI = 3.14159;
        double radius, area, circumference;
        Scanner scanner;

        DecimalFormat df = new DecimalFormat("0.000");
        scanner = new Scanner(System.in);

        //Get input
        System.out.print("Enter radius: ");
        radius = scanner.nextDouble();

        //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));
    }
}
```

Scanner class is used to teach students console-based input

## Student Pedagogy

### Design Guidelines



**Always define a constructor and initialize data members fully in the constructor so an object will be created in a valid state.**

### Design Guidelines

provide tips on good program design.

**Helpful Reminders**

provide tips for students to help them remember key concepts

**Helpful Reminder**

List the **catch** blocks in the order of specialized to more general exception classes. At most one **catch** block is executed, and all other **catch** blocks are ignored.

**Take my Advice**

It is not necessary to create an object for every variable we use. Many novice programmers often make this mistake. For example, we write

```
Fraction f1, f2;
f1 = new Fraction(24, 36);
f2 = f1.simplify();
```

We didn't write

```
Fraction f1, f2;
f1 = new Fraction(24, 36);
f2 = new Fraction(1, 1); //not necessary

f2 = f1.simplify();
```

because it is not necessary. The **simplify** method returns a **Fraction** object, and in the calling program, all we need is a name we can use to refer to this returned **Fraction** object. Don't forget that the object name (variable) and the actual object instance are two separate things.

**Take My Advice** boxes give students advice on learning effective programming.

**You Might Want to Know**

boxes give students interesting bits of information.

**You Might Want to Know**

We can turn our simulation program into a real one by replacing the **Door** class with a class that actually controls the door. Java provides a mechanism called Java Native Interface (JNI) which can be used to embed a link to a low-level device driver code, so calling the **open** method actually unlocks the door.

**Quick Check**

1. What's wrong with the following code? Identify all errors.

```
try {
    number = Integer.parseInt("123");
    if (num > 100) {
        catch new Exception("Out of bound");
    }
} catch {
    System.out.println("Cannot convert to int");
} finally (Exception e) {
    System.out.println("Always print");
}
```

**Quick Check**

exercises at the end of sections allow students to test their comprehension of topics.

## Supplements for Instructors and Students

On-Line Learning Center is located at [www.mhhe.com/wu](http://www.mhhe.com/wu)

The screenshot shows the 'Information Center' for the book. It includes a navigation menu on the left with links like 'About the Author', 'Book Preface', 'Sample Chapter', 'Table of Contents', 'What's New', 'Feature Summary', 'Supplements', 'PageOut', and 'Reviewer Quotes'. The main content area provides details about the book, including the author's name (C. Thomas Wu), ISBN (0072946520), and copyright year (2006). It also contains a paragraph describing the book's approach to teaching Java and a section titled 'In the fourth edition...' which highlights changes in class definition and Java 1.5 features. A 'LearningCenter' logo is visible on the left side of the page.

### For Instructors

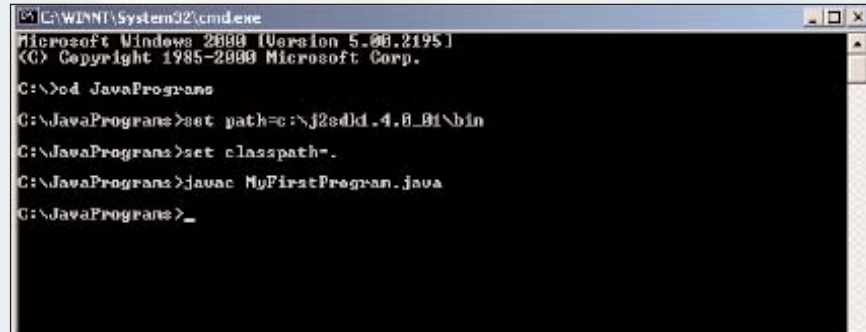
- Complete set of **PowerPoints**, including lecture notes and figures.

The slide is titled 'Method Declaration Elements'. It shows a Java method declaration: `public static void main( String[ ] args ) {`. Above the code, five boxes labeled 'Modifier', 'Modifier', 'Return Type', 'Method Name', and 'Parameter' have arrows pointing to their respective parts in the code. Below the opening curly brace, a dashed box labeled 'Method Body' contains the following code: `JFrame myWindow;  
myWindow = new JFrame( );  
myWindow.setSize(300, 200);  
myWindow.setTitle("My First Java Program");  
myWindow.setVisible(true);`. The slide footer includes the text '©2004 McGraw-Hill Higher Education' and '4th Ed Chapter 2-21'.

- **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.



```
C:\WINNT\System32\cmd.exe
Microsoft Windows [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>cd JavaPrograms
C:\JavaPrograms>set path=c:\jdk1.4.0_01\bin
C:\JavaPrograms>set classpath=.
C:\JavaPrograms>javac MyFirstProgram.java
C:\JavaPrograms>_
```

- **Interactive Quizzes** allow students to test what they learn and get immediate feedback.
- **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.

### Acknowledgments

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

#### Wu Focus Group—Jackson Hole, WY

**Elizabeth Adams**, *James Madison University*

**GianMario Besana**, *DePaul University*

**Michael Buckley**, *State University of New York, Buffalo*

**James Cross**, *Auburn University*

**Priscilla Dodds**, *Georgia Perimeter College*

**Christopher Eliot**, *University of Massachusetts-Amherst*

**Joanne Houlahan**, *John Hopkins University*

**Len Myers**, *California Polytechnic State University, San Luis Obispo*

**Hal Perkins**, *University of Washington*

**William Shea**, *Kansas State University*

**Marge Skubic**, *University of Missouri, Columbia*

**Bill Sverdlik**, *Eastern Michigan University*

**Suzanne Westbrook**, *University of Arizona*

**Reviewers**

**Ajith, Abraham**, *Oklahoma State University*  
**Elizabeth Adams**, *James Madison University*  
**David L. Atkins**, *University of Oregon*  
**GianMario Besana**, *DePaul University*  
**Robert P. Burton**, *Brigham Young University*  
**Michael Buckley**, *State University of New York, Buffalo*  
**Rama Chakrapani**, *Tennessee Technological University*  
**Teresa Cole**, *Boise State University*  
**James Cross**, *Auburn University*  
**Priscilla Dodds**, *Georgia Perimeter College*  
**Kossi Delali Edoh**, *Montclair State University*  
**Christopher Eliot**, *University of Massachusetts-Amherst*  
**Michael Floeser**, *Rochester Institute of Technology*  
**Joanne Houlahan**, *John Hopkins University*  
**Michael N. Huhns**, *University of South Carolina*  
**Eliot Jacobson**, *University of California, Santa Barbara*  
**Martin Kendall**, *Montgomery Community College*  
**Mike Litman**, *Western Illinois University*  
**Len Myers**, *California Polytechnic State University, San Luis Obispo*  
**Jun Ni**, *University of Iowa*  
**Robert Noonan**, *College of William and Mary*  
**Jason S. O'Neal**, *Mississippi College*  
**Hal Perkins**, *University of Washington*  
**Gerald Ross**, *Lane Community College*  
**William Shea**, *Kansas State University*  
**Jason John Schwarz**, *North Carolina State University*  
**Marge Skubic**, *University of Missouri, Columbia*  
**Bill Sverdlik**, *Eastern Michigan University*  
**Peter Stanchev**, *Kettering University*  
**Krishnaprasad Thirunarayan**, *Wright State University*  
**David Vineyard**, *Kettering University*  
**Suzanne Westbrook**, *University of Arizona*  
**Melissa Wiggins**, *Mississippi College*  
**Zhiguang Xu**, *Valdosta State University*

**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, we continue to publish the book under my former name. For those who care to find out the reasons for changing my name (they are not dramatic) can do so by visiting my website ([www.drcaffeine.com](http://www.drcaffeine.com)).