

# **Java Programming Second Edition (Updated)**

Instructor's Edition

---

PREVIEW

**NOT FOR PRINTING OR INSTRUCTIONAL USE**

# Java Programming, Second Edition (Updated)

---

**Series Product Managers:** Caryl Bahner-Guhin and Adam A. Wilcox  
**Developmental Editor:** Linda Long  
**Project Editor:** Kim Neitzka  
**Series Designer:** Adam A. Wilcox

COPYRIGHT © 2004 Axzo Press

ALL RIGHTS RESERVED. No part of this work may be reproduced, transcribed, or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, Web distribution, or information storage and retrieval systems—without the prior written permission of the publisher.

For more information, go to [www.axzopress.com](http://www.axzopress.com).

## **Trademarks**

ILT Series is a trademark of Axzo Press.

Some of the product names and company names used in this book have been used for identification purposes only and may be trademarks or registered trademarks of their respective manufacturers and sellers.

## **Disclaimer**

We reserve the right to revise this publication and make changes from time to time in its content without notice.

ISBN-10: 0-619-28726-8 / ISBN-13: 978-0-619-28726-9

Printed in the United States of America

1 2 3 4 5 GL 06 05 04 03

**NOT FOR PRINTING OR INSTRUCTIONAL USE**

# Contents

<b>Introduction</b>	<b>v</b>
Topic A: About the manual.....	vi
Topic B: Setting student expectations .....	xi
Topic C: Classroom setup.....	xvi
Topic D: Support.....	xviii
<b>Getting started</b>	<b>1-1</b>
Topic A: Programming basics.....	1-2
Topic B: The Java platform .....	1-7
Topic C: Installing Java .....	1-10
Topic D: Language syntax and conventions .....	1-16
Unit summary: Getting started .....	1-34
<b>Using data in a program</b>	<b>2-1</b>
Topic A: Constants and variables .....	2-2
Topic B: Data types and character sets .....	2-7
Unit summary: Using data in a program .....	2-30
<b>Methods, classes, and objects</b>	<b>3-1</b>
Topic A: Methods .....	3-2
Topic B: Classes .....	3-18
Unit summary: Methods, classes, and objects.....	3-40
<b>Advanced object concepts</b>	<b>4-1</b>
Topic A: Blocks and scope .....	4-2
Topic B: Method overloading.....	4-8
Topic C: Constants .....	4-24
Topic D: Prewritten imported methods.....	4-32
Unit summary: Advanced object concepts .....	4-41
<b>Input and selection</b>	<b>5-1</b>
Topic A: Keyboard input .....	5-2
Topic B: Control flow statements .....	5-14
Topic C: Operators .....	5-35
Unit summary: Input and selection .....	5-43
<b>Loops</b>	<b>6-1</b>
Topic A: Loop structures .....	6-2
Topic B: For and nested loops .....	6-15
Unit summary: Loops.....	6-23
<b>Characters and strings</b>	<b>7-1</b>
Topic A: Strings.....	7-2
Topic B: The StringBuffer class .....	7-16
Unit summary: Characters and strings .....	7-22
<b>Arrays</b>	<b>8-1</b>
Topic A: Introduction to arrays.....	8-2

Topic B: Arrays of objects .....	8-10
Topic C: Searching an array .....	8-17
Unit summary: Arrays .....	8-25
<b>Array manipulations</b>	<b>9-1</b>
Topic A: Manipulating an array .....	9-2
Topic B: Sorting arrays .....	9-19
Topic C: Two-dimensional arrays .....	9-31
Unit summary: Array manipulations .....	9-35
<b>Applets</b>	<b>10-1</b>
Topic A: Applets and HTML documents .....	10-2
Topic B: Applets with Swing components .....	10-9
Unit summary: Applets .....	10-21
<b>Event-driven programming</b>	<b>11-1</b>
Topic A: Event-driven programming in applets .....	11-2
Topic B: Life cycle of a Swing applet .....	11-16
Topic C: Enhancing a Swing applet .....	11-25
Unit summary: Event-driven programming .....	11-36
<b>Graphics</b>	<b>12-1</b>
Topic A: Basic graphic methods .....	12-2
Topic B: More about graphics .....	12-11
Topic C: Drawing objects .....	12-36
Topic D: Adding sound, images, and animations .....	12-46
Unit summary: Graphics .....	12-58
<b>Introduction to inheritance</b>	<b>13-1</b>
Topic A: Inheritance .....	13-2
Topic B: Superclasses .....	13-13
Topic C: Information hiding and protection .....	13-30
Unit summary: Introduction to inheritance .....	13-39
<b>Advanced inheritance concepts</b>	<b>14-1</b>
Topic A: Abstract classes .....	14-2
Topic B: Array of objects and comparing objects .....	14-15
Topic C: Interfaces and packages .....	14-24
Unit summary: Advanced inheritance concepts .....	14-30
<b>Swing components</b>	<b>15-1</b>
Topic A: Frames .....	15-2
Topic B: Swing event listeners .....	15-11
Unit summary: Swing components .....	15-38
<b>Layout managers and events</b>	<b>16-1</b>
Topic A: Layout managers and JPanel .....	16-2
Topic B: Events and event handling .....	16-22
Unit summary: Layout managers and events .....	16-40
<b>Exception handling</b>	<b>17-1</b>
Topic A: Exceptions .....	17-2
Topic B: Error handling methods .....	17-18

Topic C: Call stack and user-defined exceptions.....	17-32
Unit summary: Exception handling.....	17-39
<b>File input and output</b>	<b>18-1</b>
Topic A: Files and streams .....	18-2
Topic B: Input and output.....	18-16
Unit summary: File input and output .....	18-42
<b>Multithreading and animations</b>	<b>19-1</b>
Topic A: Multithreading .....	19-2
Topic B: Animations.....	19-19
Unit summary: Multithreading and animations.....	19-49
<b>Course summary</b>	<b>S-1</b>
Topic A: Course summary .....	S-2
Topic B: Continued learning after class .....	S-6
<b>Glossary</b>	<b>G-1</b>
<b>Index</b>	<b>I-1</b>

PREVIEW

**NOT FOR PRINTING OR INSTRUCTIONAL USE**

# Java Programming Second Edition (Updated)

## Introduction

After reading this introduction, you will know how to:

- A** Use ILT Series training manuals in general.
- B** Use prerequisites, a target student description, course objectives, and a skills inventory to properly set students' expectations for the course.
- C** Set up a classroom to teach this course.
- D** Get support for setting up and teaching this course.

## Topic A: About the manual

### ILT Series philosophy

Our goal is to make you, the instructor, as successful as possible. To that end, our training manuals facilitate students' learning by providing structured interaction with the software itself. While we provide text to help you explain difficult concepts, the hands-on activities are the focus of our courses. Leading the students through these activities will teach the skills and concepts effectively.

We believe strongly in the instructor-led classroom. For many students, having thinking, feeling instructor in front of them will always be the most comfortable way to learn. Because the students' focus should be on you, our manuals are designed and written to facilitate your interaction with the students, and not to call attention to manuals themselves.

We believe in the basic approach of setting expectations, then teaching, and providing summary and review afterwards. For this reason, lessons begin with objectives and end with summaries. We also provide overall course objectives and a course summary to provide both an introduction to and closure on the entire course.

Our goal is your success. We encourage your feedback in helping us to continually improve our manuals to meet your needs.

### Manual components

The manuals contain these major components:

- Table of contents
- Introduction
- Units
- Course summary
- Glossary
- Index

Each element is described below.

#### Table of contents

The table of contents acts as a learning roadmap for you and the students.

#### Introduction

The introduction contains information about our training philosophy and our manual components, features, and conventions. It contains target student, prerequisite, objective, and setup information for the specific course. Finally, the introduction contains support information.



**Units**

Units are the largest structural component of the actual course content. A unit begins with a title page that lists objectives for each major subdivision, or topic, within the unit. Within each topic, conceptual and explanatory information alternates with hands-on activities. Units conclude with a summary comprising one paragraph for each topic, and an independent practice activity that gives students an opportunity to practice the skills they've learned.

The conceptual information takes the form of text paragraphs, exhibits, lists, and tables. The activities are structured in two columns, one telling students what to do, the other providing explanations, descriptions, and graphics. Throughout a unit, instructor notes are found in the left margin.

**Course summary**

This section provides a text summary of the entire course. It is useful for providing closure at the end of the course. The course summary also indicates the next course in this series, if there is one, and lists additional resources students might find useful as they continue to learn about the software.

**Glossary**



The glossary provides definitions for all of the key terms used in this course.

**Index**

The index at the end of this manual makes it easy for you and your students to find information about a particular software component, feature, or concept.

## Manual conventions

We've tried to keep the number of elements and the types of formatting to a minimum in the manuals. We think this aids in clarity and makes the manuals more classically elegant looking. But there are some conventions and icons you should know about.

<i>Instructor note/icon</i>	<b>Convention</b>	<b>Description</b>
	<i>Italic text</i>	In conceptual text, indicates a new term or feature.
	<b>Bold text</b>	In unit summaries, indicates a key term or concept. In an independent practice activity, indicates an explicit item that you select, choose, or type.
	Code font	Indicates code or syntax.
	Longer strings of code will look like this. <code>▶</code> <code>▶</code> <code>▶</code>	In the hands-on activities, any code that's too long to fit on a single line is divided into segments by one or more continuation characters (▶). This code should be entered as a continuous string of text.
<i>Instructor notes.</i>		In the left margin, provide tips, hints, and warnings for the instructor.
	Select <b>bold item</b>	In the left column of hands-on activities, bold sans-serif text indicates an explicit item that you select, choose, or type.
	Keycaps like 	Indicate a key on the keyboard you must press.
 <i>Warning icon.</i>		Warnings prepare instructors for potential classroom management problems.
 <i>Tip icon.</i>		Tips give extra information the instructor can share with students.
 <i>Setup icon.</i>		Setup notes provide a realistic business context for instructors to share with students, or indicate additional setup steps required for the current activity.
 <i>Projector icon.</i>		Projector notes indicate that there is a PowerPoint slide for the adjacent content.

## Hands-on activities

The hands-on activities are the most important parts of our manuals. They are divided into two primary columns. The “Here’s how” column gives short directions to the students. The “Here’s why” column provides explanations, graphics, and clarifications. To the left, instructor notes provide tips, warnings, setups, and other information for the instructor only. Here’s a sample:

*Do it!*

*Take the time to make sure your students understand this worksheet. We'll be here a while.*

### A-1: Creating a commission formula

Here's how	Here's why
1 Open Sales	This is an oversimplified sales compensation worksheet. It shows sales totals, commissions, and incentives for five sales reps.
2 Observe the contents of cell F4	<div style="border: 1px solid black; padding: 2px; display: inline-block;"> <span style="border: 1px solid black; padding: 2px;">F4</span> <span style="border: 1px solid black; padding: 2px;">=</span> <span style="border: 1px solid black; padding: 2px;">=E4*C_Rate</span> </div> <p>The commission rate formulas use the name “C_Rate” instead of a value for the commission rate.</p>

For these activities, we have provided a collection of data files designed to help students learn each skill in a real-world business context. As students work through the activities, they will modify and update these files. Of course, they might make a mistake and, therefore, want to re-key the activity starting from scratch. To make it easy to start over, students will rename each data file at the end of the first activity in which the file is modified. Our convention for renaming files is to add the word “My” to the beginning of the file name. In the above activity, for example, students are using a file called “Sales” for the first time. At the end of this activity, they would save the file as “My sales,” thus leaving the “Sales” file unchanged. If students make mistakes, they can start over using the original “Sales” file.

In some activities, however, it may not be practical to rename the data file. Such exceptions are indicated with an instructor note. If students want to retry one of these activities, you will need to provide a fresh copy of the original data file.

## PowerPoint presentations

Each unit in this course has an accompanying PowerPoint presentation. These slide shows are designed to support your classroom instruction while providing students with a visual focus. Each one begins with a list of unit objectives and ends with a unit summary slide. We strongly recommend that you run these presentations from the instructor's station as you teach this course. A copy of PowerPoint Viewer is included, so it is not necessary to have PowerPoint installed on your computer.

### The ILT Series PowerPoint add-in

The CD also contains a PowerPoint add-in that enables you to do two things:

- Create slide notes for the class
- Display a control panel for the Flash movies embedded in the presentations

To load the PowerPoint add-in:

- 1 Copy the Course\_ILT.ppa file to a convenient location on your hard drive.
- 2 Start PowerPoint.
- 3 Choose Tools, Macro, Security to open the Security dialog box. On the Security Level tab, select Medium (if necessary), and then click OK.
- 4 Choose Tools, Add-Ins to open the Add-Ins dialog box. Then, click Add New.
- 5 Browse to and select the Course\_ILT.ppa file, and then click OK. A message box will appear, warning you that macros can contain viruses.
- 6 Click Enable Macros. The Course\_ILT add-in should now appear in the Available Add-Ins list (in the Add-Ins dialog box). The "x" in front of Course\_ILT indicates that the add-in is loaded.
- 7 Click Close to close the Add-Ins dialog box.

After you complete this procedure, a new toolbar will be available at the top of the PowerPoint window. This toolbar contains a single button labeled "Create SlideNotes." Click this button to generate slide notes files in both text (.txt) and Excel (.xls) format. By default, these files will be saved to the folder that contains the presentation. If the PowerPoint file is on a CD-ROM or in some other location to which the SlideNotes files cannot be saved, you will be prompted to save the presentation to your hard drive and try again.

When you run a presentation and come to a slide that contains a Flash movie, you will see a small control panel in the lower-left corner of the screen. You can use this panel to start, stop, and rewind the movie, or to play it again.

## Topic B: Setting student expectations

Properly setting students' expectations is essential to your success. This topic will help you do that by providing:

- Prerequisites for this course
- A description of the target student at whom the course is aimed
- A list of the objectives for the course
- A skills assessment for the course

### Course prerequisites

Students taking this course should be familiar with personal computers and the use of a keyboard and a mouse. Furthermore, this course assumes that students have no programming experience.

### Target student

The target students are programmers or Web developers who want to use Java. The students will be able to develop applications and applets by using the Java programming language. They will also be able to build visually interesting GUI and Web-based applications.

### Course objectives

You should share these overall course objectives with your students at the beginning of the day. This will give the students an idea about what to expect, and will also help you identify students who might be misplaced. Students are considered misplaced when they lack the prerequisite knowledge or when they already know most of the subject matter to be covered.

After completing this course, students will know how to:

- Describe the basic and object-oriented programming concepts, the Java platform, and types of Java programs, as well as how to install and configure the Java SDK.
- Use constants, variables, and various data types.
- Create and use methods, classes, and instantiate objects from classes.
- Define blocks and scope of a variable, overload methods and constructors, and work with constants.
- Accept keyboard input, use the `JOptionPane` class, draw flowcharts and make decisions by using `if...else` and `switch` statements, use `AND`, `OR`, conditional, and `NOT` operators, and their order of precedence.
- Use `while` loops, `do...while` loops, shortcut arithmetic operators, for loops and, nested loops.
- Manipulate characters and use `String` methods and the `StringBuffer` class.
- Declare, initialize an array, declare and create an array of objects, and search an array.

- Pass arrays to methods, use the length field, create an array of strings, sort arrays, and create two-dimensional and multidimensional arrays.
- Create HTML documents to run applets containing AWT components and create applets with Swing components.
- Define event-driven programming and handle events with Swing components, identify the life cycle of an applet, and enhance Swing applets.
- Implement basic graphic methods, draw lines, rectangles, ovals, arcs and polygons, and add sound, images, and animations to Swing applets.
- Implement inheritance and extend classes, override superclass methods, and use information hiding and protection.
- Create abstract classes and use dynamic method binding, create an array of objects, compare objects, and create interfaces and packages.
- Use the JFrame and JPanel class, use the Swing event listeners and Swing components.
- Use layout managers, JPanel and AWTEvent class methods, and handle mouse events.
- Throw and catch exceptions, specify and handle exceptions, and trace and create your own exceptions.
- Use the File class and streams; write to and read from a file.
- Implement multithreading, create and use animations.

## Skills inventory

Use the following form to gauge students' skill level entering the class (students have copies in the introductions of their student manuals). For each skill listed, have students rate their familiarity from 1 to 5, with five being the most familiar. Emphasize that this is not a test. Rather, it is intended to provide students with an idea of where they're starting from at the beginning of class. If a student is wholly unfamiliar with all the skills, he or she might not be ready for the class. A student who seems to understand all of the skills, on the other hand, might need to move on to the next course in the series.

Skill	1	2	3	4	5
Describing basic and object-oriented programming concepts					
Describing the Java platform and types of Java programs					
Installing and configuring the Java SDK					
Writing a simple Java program					
Using constants and variables					
Using various data types					
Creating and using methods					
Creating classes and instantiating objects from classes					
Defining blocks and scope of a variable					
Overloading methods and constructors					
Working with constants					
Accepting keyboard input and using the JOptionPane class					
Drawing flowcharts and making decisions by using if...else and switch statements					
Using AND, OR, conditional, and NOT operators, and their order of precedence					
Using while and do...while loops, and discussing shortcut arithmetic operators					
Using for and nested loops					
Manipulating characters and using String methods					
Using the StringBuffer class					

<b>Skill</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
Declaring and initializing an array					
Declaring and creating an array of objects					
Searching an array					
Passing arrays to methods, using the length field, and creating arrays of strings					
Sorting arrays					
Creating two-dimensional and multidimensional arrays					
Creating an HTML document to run an applet containing AWT components					
Creating an applet with Swing components					
Defining event-driven programming and handling events with Swing components					
Identifying the life cycle of an applet					
Enhancing the Swing applet					
Implementing basic graphic methods					
Creating various graphic objects					
Drawing lines, rectangles, ovals, arcs and polygons					
Adding sound, images, and animations to Swing applets					
Implementing inheritance and extend classes					
Overriding superclass methods					
Using information hiding and protection					
Creating abstract classes and use dynamic method binding					
Creating an array of objects and comparing objects					
Creating interfaces and packages					
Using the JFrame and JPanel class					
Using the Swing event listeners and Swing components					



<b>Skill</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
Use layout managers and JPanel					
Using AWTEvent class methods and handling mouse events					
Throwing and catching exceptions					
Specifying and handling exceptions					
Tracing and creating your own exceptions					
Using the File class and streams					
Writing to and reading from a file					
Implementing multithreading					
Creating and using animations					

## Topic C: Classroom setup

All our courses assume that each student has a personal computer to use during the class. Our hands-on approach to learning requires they do. This topic gives information on how to set up the classroom to teach this course. It includes minimum requirements for the students' personal computers, setup information for the first time you teach the class, and setup information for each time that you teach after the first time you set up the classroom.

### Student computer requirements

Each student's personal computer should have:

#### Hardware

- A keyboard and a mouse
- A Pentium II-class processor or higher
- 128MB of RAM
- At least 4GB hard disk space
- 1.44 MB 3½-inch floppy disk drive
- CD-ROM drive supported by Windows 2000
- A monitor capable of 256 colors at 800×600 resolution
- Internet access, if you want to download the Student Data files from [www.courseilt.com/instructor\\_tools.html](http://www.courseilt.com/instructor_tools.html)

#### Software

- Windows 2000 Professional (with Service Pack 2 or later), Microsoft Windows NT (with Service Pack 5 or later), or Windows XP
- Java 2 SDK v1.4.1 or later
- Internet Explorer 6.0 or later
- Notepad

### First-time setup instructions

The first time you teach this course, you will need to perform the following steps to set up each student computer.

- 1 Install Microsoft Windows 2000 Professional according to the software manufacturer's instructions. You can also use Microsoft Windows XP Professional. Screen shots in this course were taken using Microsoft Windows XP Professional and students' screens might look somewhat different.
- 2 Install Java 2 SDK v1.4.1 according to the software manufacturer's instructions.
- 3 Install Internet Explorer 6.0.
- 4 Install all other software mentioned in the software section as per the manufacturer's instructions.

- 5 Download the Student Data files for the course. You can download the data directly to student machines, to a central location on your own network, or to a disk.
  - a Connect to [www.courseilt.com/instructor\\_tools.html](http://www.courseilt.com/instructor_tools.html).
  - b Click the link for Java/J++ to display a page of course listings, and then click the link for Java Programming, Second Edition (Updated).
  - c Click the link for downloading the Student Data files, and follow the instructions that appear on your screen.

### **Setup instructions for every class**

Every time you teach this course, you will need to perform the following steps to set up each student computer.

- 1 Delete the contents of the Student Data folder, if necessary. (If this is the first time you are teaching the course, create a folder named Student Data at the root of the hard drive.)
- 2 Copy the data files to the Student Data folder. (See the download instructions in the preceding section.)

## Topic D: Support

Your success is our primary concern. If you need help setting up this class or teaching a particular unit, topic, or activity, please don't hesitate to get in touch with us.

### Contacting us

Please contact us through our Web site, [www.axzopress.com](http://www.axzopress.com). You will need to provide the name of the course, and be as specific as possible about the kind of help you need.

### Instructor's tools

Our Web site provides several instructor's tools for each course, including course outlines and answers to frequently asked questions. To download these files, go to [www.axzopress.com](http://www.axzopress.com). Then, under Downloads, click Instructor-Led Training and browse our subject categories.

# Unit 1

## Getting started

**Unit time: 75 minutes**

Complete this unit, and you will know how to:

- A** Describe basic and object-oriented programming concepts.
- B** Describe the Java platform and identify types of Java programs.
- C** Install and configure the Java SDK.
- D** Write a simple Java program and add comments.

## Topic A: Programming basics

### Explanation

Before you learn about the Java programming language, you must have a clear understanding of basic programming—concepts, specifically, object-oriented programming concepts.

### Programming concepts

A computer *program* is a set of instructions that you write to tell a computer what to do. Computers are constructed from circuitry consisting of small on/off switches. You can write a computer program by writing something similar to the lines below:

```
first switch-on
second switch-off
third switch-off
fourth switch-on
```

Your program could continue for several thousand switches. A program written in this style is written in *machine language*, the most basic circuitry-level language. The main problem with this approach lies in keeping track of the many switches involved in programming a task. Another problem with this approach includes difficulties in discovering the errant switch or switches if the program does not operate as expected. Additionally, the number and location of switches varies from computer to computer, which means that you would need to customize a machine language program for every type of machine on which you want the program to run.

Fortunately, programming has evolved into an easier task because of the development of high-level programming languages. A high-level programming language allows you to use a vocabulary of reasonable terms such as “read,” “write,” or “add,” instead of the sequences of on and off switches that perform these tasks. High-level languages also allow you to assign intuitive names to areas of computer memory, such as “hoursWorked” or “rateOfPay,” rather than having to remember the memory locations (switch numbers) of those values.

Each high-level language has its own *syntax*, or rules of the language. For example, depending on the language, you might use the verb “print” or “write” to produce output. All languages have a specific, limited vocabulary and a specific set of rules for using that vocabulary.

Programmers use a computer program called a *compiler* to translate their high-level language statements into machine code. The compiler issues an error message each time the programmer uses the programming language incorrectly. Subsequently, the programmer can correct the error and attempt another translation by re-compiling the program. Some languages use an *interpreter* to read the compiled code line by line at run time. Java programs are both compiled and interpreted.

A Java interpreter can run either as a stand-alone program or as part of a Web browser, such as Netscape Navigator or Microsoft Internet Explorer. When the Java interpreter runs in a Web browser, it can be invoked automatically to run applets in a Web page. The ability to run applets in a Web browser is one feature that distinguishes Java from other programming languages.

When you are learning a computer programming language, such as the Java programming language, C++, or Visual Basic, you need to learn the vocabulary and syntax rules for that language. In addition to learning the correct syntax for a particular language, a programmer must also understand computer programming logic.

The *logic* behind any program involves executing the various statements and procedures in the correct order to produce the desired results. For example, you would not write statements to tell the computer program to process data until the program had properly read the data. Similarly, you might be able to use a computer language's syntax correctly, but be unable to execute a logically constructed, workable program. Examples of logical errors include multiplying two values when you meant to divide them, or producing output prior to obtaining the appropriate input.

Do it!

### A-1: Discussing programming basics

#### Questions and answers

1 How do high-level languages make programming easier?

*By allowing the programmer to use a vocabulary of intuitive terms.*

2 What defines the vocabulary rules of a programming language?

*Syntax*

3 What does computer programming logic refer to?

*The logic behind any program involves executing the various statements and procedures in the correct order to produce the desired results.*

4 Is Java source code compiled or interpreted?

*Both. Java is first compiled, and then interpreted.*

5 What is one feature that distinguishes Java from other programming languages?

*The ability to run applets in a Web browser*

## Procedural and object-oriented programming

Explanation

There are two popular approaches to writing computer programs: procedural programming and object-oriented programming.

### Procedural programming

In *procedural programming*, the individual operations used in a computer program are often grouped into logical units called *procedures*. In a procedural program, one procedure follows another from beginning to the end. You write all of the procedures for a program, and then when the program executes, each procedure takes place in order based on the program's logic.

One example of a procedural program is a series of four or five comparisons and calculations that together determine an individual's federal withholding tax value. These comparisons and calculations might be grouped as a procedure named `calculateFederalWithholding`. A procedural program defines the variable memory locations, and then calls a series of procedures to input, manipulate, and output the values stored in those locations. A single procedural program often contains hundreds of variables and thousands of procedure calls.

### Object-oriented programming

*Object-oriented programming* is an extension of procedural programming that takes a slightly different approach to writing computer programs. Writing object-oriented programs involves both creating objects and creating applications that use those objects. Objects often interrelate with other objects, and once created, objects can be reused to develop new programs. Thinking in an object-oriented manner involves envisioning program components as objects that are similar to concrete objects in the real world; then you can manipulate the objects to achieve a desired result.

If you have ever used a computer that uses a command-line operating system (such as DOS), and if you have also used a graphical user interface, or GUI (such as Windows), then you are familiar with the difference between procedural and object-oriented programs. If you want to move several files from a floppy disk to a hard disk, you can use either a typed command at a prompt or command line, or you can use a mouse in a graphical environment to accomplish the task. The difference lies in whether you issue a series of commands in sequence to move the three files, or if you drag icons representing the files from one screen location to another, much as you would physically move paper files from one file cabinet to another in your office. You can move the same three files using either operating system, but the GUI system allows you to manipulate the files like their real-world paper counterparts. In other words, the GUI system allows you to treat files as objects.

### Object-oriented concepts

*Objects* both in the real world and in object-oriented programming are made up of states and methods. The states of an object are commonly referred to as its attributes.

*Attributes* are the characteristics that define an object as part of a class. For example, some of your automobile's attributes are its make, model, year, and purchase price. Other attributes include whether the automobile is currently running, its gear, its speed, and whether it is dirty. All automobiles possess the same attributes but not, of course, the same values for those attributes. Similarly, your dog has the attributes of its breed, name, age, and whether his or her shots are current.



The term *class* describes a group or collection of objects with common properties. An existing object of a class is called an *instance*. Therefore, your red Chevrolet sedan with the dent can be considered an instance of the class that is made up of all automobiles, and your golden retriever named Goldie is an instance of the class that is made up of all dogs. Thinking of items as instances of a class allows you to apply your general knowledge of the class to individual members of the class. A particular instance of an object takes on, or *inherits*, its attributes from a more general category. If a general class Dog has defined attributes and methods, they can be passed on to a specific class Golden Retriever Dog with minimal programming effort. If your friend purchases an automobile, you know it has a model name, and if your friend gets a dog, you know the dog has a breed. You might not know the exact contents of your friend's automobile, its current state or her automobile's speed or her dog's shots, but you do know what attributes exist for the Automobile and Dog classes. Similarly, in a GUI operating environment, you expect each component to have specific, consistent attributes, such as a menu bar and a title bar, because each component inherits these attributes as a member of the general class of GUI components.

Programmers using Java begin their class names with an uppercase letter. Thus, the class that defines the attributes and methods of an automobile would probably be named Automobile, and the class for dogs would probably be named Dog. However, following this convention is not required to produce a workable program.

Besides attributes, objects can use methods to accomplish tasks. A *method* is a self-contained block of program code. Automobiles, for example, can move forward and backward. They can also be filled with gasoline or be washed, both of which can be programmed as methods to change some of their attributes. Methods exist for ascertaining certain attributes, such as the current speed of an automobile and the current status of its gas tank. Similarly, a dog can walk or run, eat food, and get a bath, and there are methods to determine how hungry the dog is. GUI operating system components can be maximized, minimized, and dragged. Like procedural programs, object-oriented programs have variables (attributes) and procedures (methods), but the attributes and methods are encapsulated into objects that are then used much like real-world objects.

*Encapsulation* refers to the hiding of data and methods within an object. Encapsulation provides the security that keeps data and methods safe from inadvertent changes. Programmers sometimes refer to encapsulation as using a "black box," or a device that you can use without regard to the internal mechanisms. A programmer gets to access and use the methods and data contained in the black box but cannot change them.

If an object's methods are well written, the user is unaware of the low-level details of how the methods are executed, and the user must simply understand the interface or interaction between the method and the object. For example, if you can fill your automobile with gasoline, it is because you understand the interface between the gas pump nozzle and the vehicle's gas tank opening. You do not need to understand how the pump works mechanically or where the gas tank is located inside your vehicle. If you can read your speedometer, it does not matter how the displayed figure is calculated. As a matter of fact, if someone produces a superior, more accurate speed-determining device and inserts it in your automobile, you do not have to know or care how it operates, as long as your interface remains the same. The same principles apply to well-constructed objects used in object-oriented programs.

Do it!

## A-2: Discussing object-oriented programming

### Questions and answers

1 In procedural programming, you define the \_\_\_\_\_.

*sequences of actions or procedures*

2 How are classes and instances related?

*Classes define entities in the abstract; instances denote specific objects that conform to the class specification.*

3 What is encapsulation?

*The hiding of data and methods within an object*

## Topic B: The Java platform

### Explanation

Java technology was developed by Sun Microsystems as both an object-oriented programming language and a platform.

A *platform* is the hardware or software environment in which a program runs. Some of the most popular platforms include Windows, Linux, Solaris, and Mac. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it is a software-only platform that runs on top of other hardware-based platforms.

Java is used both for general-purpose business programs and for interactive World Wide Web-based Internet programs. Some of the advantages that have made the Java programming language so popular in recent years are its security features, and the fact that it is *architecturally neutral*, which means that you can use the Java programming language to write a program that will run on any platform or operating system.

The Java platform has two components:

- The Java application programming interface (API)
- The Java Virtual Machine (JVM)

The *Java API* is a large collection of ready-made software components that provide many useful capabilities, such as GUI widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as packages.

The JVM is the base for the Java platform and is ported onto various hardware-based platforms.

### The Java environment

Java can be run on a wide variety of computers because Java does not execute instructions on a computer directly. Instead, Java runs on a hypothetical computer known as the *Java Virtual Machine* (JVM).

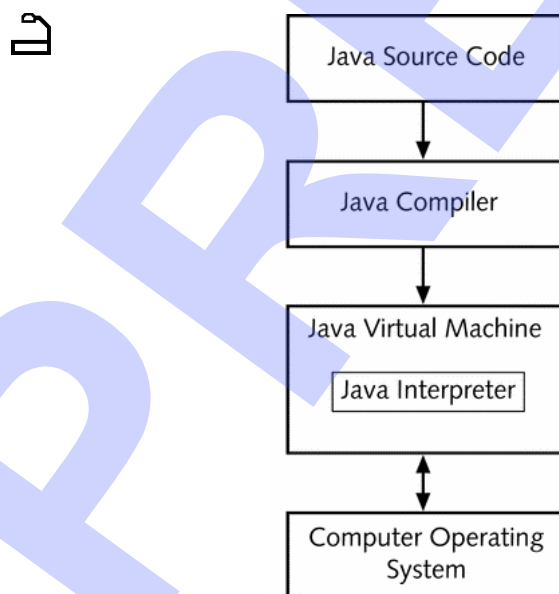


Exhibit 1-1: The Java environment

The Java environment is shown in Exhibit 1-1. You can create a Java program by using a standard text editor. The file containing the Java programming statements is also known as the *source code*. The Java compiler converts the source code into machine-independent *bytecode*. A program called the Java interpreter checks the bytecode and executes the bytecode instructions line by line within the JVM. Because the Java program is isolated from the native operating system, the Java program is insulated from the particular hardware on which it is run. Because of this insulation, the JVM provides security against intruders getting at your computer's hardware through the operating system.

In contrast, when using other programming languages, software vendors usually have to produce multiple versions of the same product—a DOS version, Windows version, Macintosh version, UNIX version, and so on—so all users can use the program. With Java, one program version will run on all these platforms. It is the Java bytecode that helps make “write once, run anywhere” possible. You can compile your program into bytecode on any platform that has a Java compiler. The bytecode can then be run on any implementation of the JVM. This means that as long as a computer has a JVM, the same program written in Java can run on Windows 2000, a Solaris workstation, or on a Mac, as shown in Exhibit 1-2.

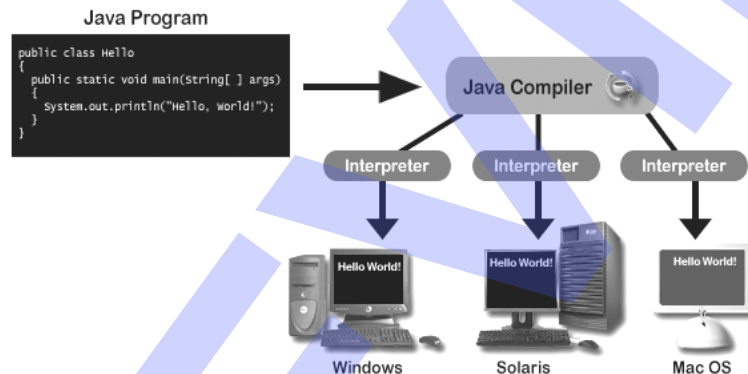


Exhibit 1-2: Java write once, run anywhere

## Program types

There are three main types of programs you can write using Java:

- *Applications* run in the JVM installed on a computer system.
- *Applets* run in the JVM built into a Web browser.
- *Servlets* run in the JVM installed on a Web server.

While applets and applications usually have some kind of user interface coupled to backend functionality, servlets only provide backend functionality. The user interface for a servlet is usually an HTML form in a browser that invokes the servlet, but any applet or application that opens a hypertext transfer protocol (HTTP) request can call a servlet.

Java applications can be further subdivided into:

- *Console applications*, which support character output to a computer screen in a DOS window
- *Windowed applications*, which create a GUI with elements such as menus, toolbars, and dialog boxes

## Advantages

Java is not as complicated as many other object-oriented languages. The Java programming language is modeled after the C++ programming language. Although neither language is “simple” to read or understand on first exposure, Java eliminates some of the most difficult-to-understand features in C++, such as pointers and multiple inheritance.

Do it!

## B-1: Discussing the Java platform

### Questions and answers

1 Java is described as being architecturally neutral; what does this mean?

*A Java program is compiled to a machine-independent bytecode. The bytecode can then be interpreted on any implementation of the JVM, so the program can run on any platform or operating system.*

2 What popular language was Java modeled after?

**C++**

3 What are the three main types of Java programs?

*Java applications, applets, and servlets*

## Topic C: Installing Java

### Explanation

Before you can develop a Java program, you will need to download and install one of the following Java 2 platforms from the Sun Microsystems Web site:

[www.java.sun.com](http://www.java.sun.com).



- **Java 2 Platform Standard Edition (J2SE):** This kit is necessary for developing all applications, except those designed for consumer devices. J2SE comes bundled with the compiler, a runtime environment, and the core APIs.
- **Java 2 Platform, Enterprise Edition (J2EE):** This packages includes an application server, Web server, J2EE APIs, support for Enterprise JavaBeans, Java Servlets API, and JavaServer Pages (JSP) technology. Use J2EE with the J2SE.
- **JavaServer Web Development Kit (JSWDK):** If you are interested in writing and testing servlets or JavaServer Pages, download this kit along with the J2SE bundle. If you want to work with Enterprise JavaBeans, then you should download the J2EE package instead.
- **Java 2 Platform, Micro Edition (J2ME):** If you are interested in developing programs for Palm Pilots, screen phones, and other consumer devices, this kit provides tools for compiling, deployment and device configuration, and APIs that are specialized for each type of device.

The Java 2 software development kits (Java SDKs) include the APIs necessary for the various types of programs you can develop with the Java programming language.

### System requirements

The Java 2 SDK is supported on the following Microsoft platforms running on Intel hardware:



- Windows 98 (1st or 2nd edition)
- Windows NT 4.0 (with Service Pack 5 or later)
- Windows ME
- Windows XP
- Windows 2000 (with Service Pack 2 or later)

**Note:** Trying to install the Java 2 SDK on a non-supported version of Windows or on a machine that does not have a sufficiently up-to-date Service Pack will cause the installer to generate the following warning:

We recommend that you do not install this Java Platform for the following reasons:

This Java Platform does not support the operating system or operating-system service pack on this machine.

A Pentium 166 MHz or faster processor with at least 32 megabytes (MB) of physical RAM is required to run graphic-based applications. Forty-eight MB of RAM is recommended for applets running within a browser using the Java Plug-in product. Running with less memory may cause disk swapping, which has a severe effect on performance. Very large programs may require more RAM for adequate performance.

You should have 120 MB of free disk space before attempting to install the Java 2 SDK software.

## Installation instructions

To install the Java 2 platform, you will run the self-installing executable to unpack and install the Java 2 SDK software bundle.

**Note:** After you install the Java 2 SDK software, you might need to reboot your system.

During installation, you must substitute the appropriate version number whenever you see the following notation:

<version number>

For example, if you download the installer for version 1.4.1\_03, the file name `j2sdk-1_4_1_<version number>-windows-i586.exe` would be `j2sdk-1_4_1_03-windows-i586.exe`.

## Installation steps

Use the following steps to install the Java 2 SDK software:

- 1 Check the download file size.

If you save the self-installing executable to disk without running it from the download page at the Java Software Web site, notice that its byte size is provided on the download page. Once the download has completed, check that you have downloaded the full, uncorrupted software file.

- 2 Uninstall any prior version of the Java 2 SDK.

If you have previously installed another version of the Java 2 SDK, uninstall it. Use the Microsoft Windows Add/Remove Programs utility, accessible from the Control Panel (Click Start, then choose Settings, and then Control Panel).

- 3 Run the Java 2 SDK installer.

**Note:** You must have administrative permissions in order to install the Java 2 SDK on Windows 2000 and XP.

The file `j2sdk-1_4_1_<version number>-windows-i586-i.exe` is the Java 2 SDK installer. If you downloaded it instead of running it directly from the Web site, double-click the installers icon. Then follow the instructions the installer provides. When done with the installation, you can delete the download file to recover disk space.

- 4 Delete the downloaded file (optional).

If you want to recover disk space, delete the file (or files) you originally downloaded.

### Installed directory tree

The Java 2 SDK has the directory structure shown in Exhibit 1-3.

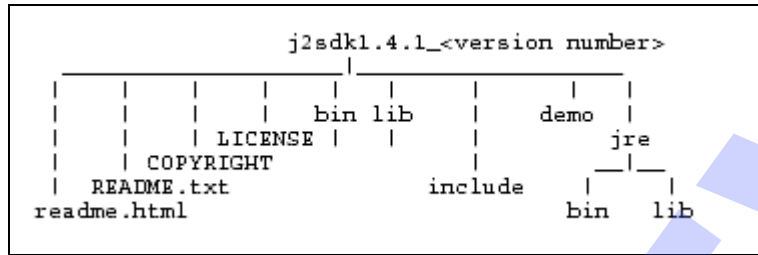


Exhibit 1-3: The Java 2 SDK directory structure

In addition, the Java Plug-in and Java Web Start will be automatically installed. Look for a Java Web Start icon on your desktop. There will also be an entry for Java Web Start in the Start menu.

Do it!

### C-1: Installing the Java 2 Platform Standard Edition

#### Questions and answers

1 What is the Web site where can you download the Java 2 platform?

***www.java.sun.com***

2 During installation, what must you do whenever you see the following notation: <version number>?

***You must substitute the appropriate update version number for the notation.***

3 What is the first step you should take after downloading the Java 2 SDK software?

***Once the download has completed, check that you have downloaded the full, uncorrupted software file.***

4 How can you recover disk space after installing the Java 2 SDK software?

***Delete the file (or files) you originally downloaded***

5 If you download and install j2sdk-1\_4\_1\_04-windows-i586-i.exe to the root of your C: drive, what would be the path to the bin directory?

***C:\j2sdk1.4.1\_04\bin***



## Setting the PATH variable

### Explanation

You can run the Java 2 SDK without setting the PATH variable. However, if you want to be able to conveniently run the Java 2 SDK executables (`javac.exe`, `java.exe`, `javadoc.exe`, etc.) from any directory without having to type the full path of the command, then you should set the PATH variable. If you do not set the PATH variable, you will need to specify the full path to the executable every time you run it, such as:

```
C:> \j2sdk1.4.1_<version number>\bin\javac First.java
```

It is useful to set the PATH permanently so it will persist after rebooting. To set the PATH permanently, you add the full path of the `j2sdk1.4.1_<version number>\bin` directory to the PATH variable. Typically this full path looks something like the following:

```
C:\j2sdk1.4.1_<version number>\bin
```

### Setting the PATH variable on Windows NT, 2000, and XP

To set the PATH permanently:

- 1 Choose Start, Settings, Control Panel, and then double-click System or System Tools.
- 2 On Windows NT, activate the Environment tab; on Windows 2000 activate the Advanced tab and then click Environment Variables.
- 3 Select Path in the User Variables and click Edit.
- 4 Add the new path value to the end of the PATH statement. A typical value for PATH is:

```
C:\j2sdk1.4.1_<version number>\bin
```

The PATH can be a series of directories separated by semi-colons (;). Windows looks for programs in the PATH directories in order, from left to right. You should have only one bin directory for a Java SDK in the path at a time (those following the first are ignored), so if one is already present, you can update it to your current version number.

- 5 Click OK.
- 6 Select Path in the System Variables and click Edit.
- 7 Add the Path value to the end of the PATH statement.
- 8 Click OK.
- 9 Click OK.
- 10 Click OK.

The new path takes effect in each new Command Prompt window you open after setting the PATH variable.

### Setting the PATH variable on Windows 98

To set the PATH permanently, open the AUTOEXEC.BAT file and add or change the PATH statement as follows:

- 1 Choose Start, Run, enter sysedit, and then click OK. The system editor opens with several windows showing. Activate the window that displays AUTOEXEC.BAT.
- 2 Look for the PATH statement. (If you do not have one, add one.) If you are not sure where to add the Path, add it to the end of the PATH statement. For example, in the following PATH statement, we've added the bin directory at the end:

```
PATH C:\WINDOWS;C:\WINDOWS\COMMAND;C:\J2SDK1.4.1_>  
      <version number>\BIN
```

- 3 To make the Path take effect in the current Command Prompt window, execute the following:

```
C:> c:\autoexec.bat
```

- 4 To find out the current value of your PATH, to see if it took effect, at the command prompt, type:

```
C:> path
```

### Setting the PATH variable on Windows ME

To set the PATH permanently:

- 1 Choose Start, Programs, Accessories, System Tools, System Information. The Microsoft Help and Support window appears.
- 2 Choose Tools, System Configuration.
- 3 Activate the Environment tab, select PATH, and click Edit.
- 4 Look for the PATH statement. (If you do not have one, add one.) If you are not sure where to add the Path, add it to the end of the PATH statement. For example, in the following PATH statement, we have added the bin directory at the end:

```
PATH C:\WINDOWS;C:\WINDOWS\COMMAND;C:\J2SDK1.4.1_>  
      <version number>\BIN
```

- 5 After you have added the location of the SDK to your PATH, save the changes and reboot your machine when prompted.

### Uninstalling the Java 2 SDK

If you want to uninstall the Java 2 SDK, use the Add/Remove Programs utility in the Windows Control Panel. As an alternative method, if you still have the original installation program that you used to install the Java 2 SDK, you can double-click it to launch an uninstall program.

*Do it!***C-2: Setting the PATH variable on Windows 2000**

<b>Here's how</b>	<b>Here's why</b>
1 Click <b>Start</b>  Choose <b>Settings, Control Panel</b>  Double-click <b>System</b>	To set PATH variables.   To open the System Properties dialog box.
2 Activate the Advanced tab  Click <b>Environment Variables</b>	To open the Environment Variables dialog box.
3 In the User variables list, select <b>Path</b>  Click <b>Edit</b>	To open the Edit User Variable dialog box. You will edit the User Path value.
4 At the end of the PATH statement, enter your Path value	A typical value for PATH is:  <code>C:\j2sdk1.4.1_&lt;version number&gt;\bin</code>  You should have only one bin directory for a Java SDK in the path at a time (those following the first are ignored), so if one is already present, you can update it to <code>j2sdk1.4.1_&lt;version number&gt;\bin</code> .
5 Click <b>OK</b>	To close the Edit User Variable dialog box.
6 In the System variables list, select <b>Path</b>  Click <b>Edit</b>	To open the Edit System Variable dialog box. You will edit the Path value.
7 At the end of the PATH statement, enter your Path value	To permanently add the Path value to the PATH statement.
8 Click <b>OK</b>	To close the Edit System Variable dialog box.
9 Click <b>OK</b>	To close the Environment Variables dialog box.
10 Click <b>OK</b>	To close the System Properties dialog box.

## Topic D: Language syntax and conventions

### Explanation

At first glance, even the simplest Java program involves a fair amount of confusing language syntax and conventions.

### Syntax basics

The best way to begin learning Java syntax is to consider a short simple program. The program shown in Exhibit 1-4 is written on seven lines, and its only task is to print “First Java program” on the screen.

```
public class First
{
    public static void main(String[] args)
    {
        System.out.println("First Java program");
    }
}
```

Exhibit 1-4: Printing a string

The following statement does the actual work in this program:

```
System.out.println("First Java program");
```

Notice that this statement ends with a semicolon. All Java programming language statements end with a semicolon.

The text “First Java program” is a *literal string* of characters; that is, it is a series of characters that will appear exactly as entered. Any literal string in Java appears between double quotation marks, as opposed to single quotation marks as in ‘First Java program’. Even though code might wrap to multiple lines, the literal string should not be broken by a hard return.

The string “First Java program” appears within parentheses because the string is an argument to a method. Arguments to methods always appear within parentheses.

*Arguments* consist of information that a method requires to perform its task. For example, you might place a catalog order with a company that sells sporting goods. Processing a catalog order is a method that consists of a set of standard procedures. However, each catalog order requires information such as which item number you are ordering and the quantity of the item desired; this information can be considered the order’s argument. If you order two of item 5432 from a catalog, you expect different results than if you order 1000 of item 9008. Likewise, if you pass the argument “Happy Holidays” to a method, you expect different results than if you pass the argument “First Java program”.

Within the statement `System.out.println("First Java program");`, the method to which you are passing “First Java program” is named `println()`. The `println()` method prints a line of output on the screen, positions the insertion point on the next line, and stands ready for additional output.

**Syntax convention:** Method names are usually followed by parentheses, as in `println()`, so you can distinguish method names from variable names.

Within the statement `System.out.println("First Java program");`, `out` is an object. The `out` object represents the screen. Several methods, including `println()`, are available with the `out` object. Of course, not all objects have a `println()` method (for instance, you can not print to a keyboard, to your automobile, or to your dog), but the creators of the Java platform assumed you would want to display output on a screen. Therefore, the `out` object was created along with a method named `println()`.

The `print()` method is similar to the `println()` method. With `println()`, after the message prints, the insertion point appears on the following line. With `print()`, the insertion point does not advance to a new line; it remains on the same line as the output.

Within the statement `System.out.println("First Java program");`, `System` is a class. Therefore, `System` defines the attributes of a collection of similar “System” objects, just as the `Dog` class defines the attributes of a collection of similar “Dog” objects. One of the `System` objects is `out`. (You can probably guess that another object is `in`, and that it represents an input device.)

**Syntax rule:** Java is case sensitive; the class named `System` is a completely different class from one named `system`, `SYSTEM`, or even `sYsTeM`.

The dots (periods) in the statement `System.out.println("First Java program");` are used to separate the names of the class, object, and method. You will use this same class-dot-object-dot-method format repeatedly in your Java programs.

The statement that prints the string “First Java program” is embedded in the program shown in Exhibit 1-4.

**Java classes**

Everything that you use within a Java program must be part of a class. When you write `public class First`, you are defining a class named `First`. You can define a Java class using any name or identifier, as long as it meets the following requirements:

- A class name must begin with a letter of the alphabet (which includes any non-English letter, such as  $\alpha$  or  $\pi$ ), an underscore, or a dollar sign.
- A class name can contain only letters, digits, underscores, or dollar signs.
- A class name cannot be a Java programming language reserved keyword, such as `public` or `class`. Exhibit 1-5 shows a list of reserved keywords in Java programming language.
- A class name cannot be one of the following values: `true`, `false`, or `null`.

abstract	double	int	strictfp
Boolean	else	interface	super
break	extends	long	switch
byte	final	native	synchronized
case	finally	new	this
catch	float	package	throw
char	for	private	throws
class	goto	protected	transient
const	if	public	try
continue	implements	return	void
default	import	short	volatile
do	instanceof	static	while

*Exhibit 1-5: Reserved keywords*

**Tip:** Java is based on Unicode, which is an international system of character representation. The term *letter* indicates English-language letters, as well as characters from Arabic, Greek, and other alphabets.

A standard convention in Java is to begin class names with an uppercase letter and employ other uppercase letters as needed to improve readability. Following are some valid and conventional class names that you can use while programming in Java:

<b>Class name</b>	<b>Description</b>
Employee	Begins with an uppercase letter
UnderGradStudent	Begins with an uppercase letter, contains no spaces, and emphasizes each new word with an initial uppercase letter
InventoryItem	Begins with an uppercase letter, contains no spaces, and emphasizes the second word with an initial uppercase letter
Budget2004	Begins with an uppercase letter and contains no spaces

You should follow established conventions for the Java programming language so your programs are easy for other programmers to interpret and follow.

Following is a list of some unconventional class names that you should not use while programming in Java:

<b>Class name</b>	<b>Why you should not use it</b>
Undergradstudent	New words are not indicated with initial uppercase letters, so the text is difficult to read.
Inventory_Item	The underscore is not commonly used to indicate new words.
BUDGET2004	The text appears as all uppercase letters.

Following is a list of some illegal class names that cannot be used while programming in Java:

<b>Class name</b>	<b>Why you cannot use it</b>
An employee	The space character is illegal.
Inventory Item	The space character is illegal.
Class	Class is a reserved word.
2001Budget	Class names cannot begin with a digit.
phone#	The # symbol is not allowed.

In the program shown in Exhibit 1-4, the line `public class First` contains the keyword `class`, which identifies `First` as a class. The reserved word `public` is an access modifier.

An *access modifier* defines the circumstances under which a class can be accessed. Public access is the most liberal type of access.

You enclose the contents of all classes within curly braces (`{` and `}`). A class can contain any number of data items and methods. In the program, the class `First` contains only one method within its curly braces. The name of the method is `main()`, and the `main()` method contains its own set of parentheses and only one statement—the `println()` statement.

**Note:** In general, white space is optional in Java. White space is any combination of spaces, tabs, and carriage returns (blank lines). However, you cannot use white space within any identifier or keyword. You can insert white space between words or lines in your program code by typing spaces, tabs, or blank lines because the compiler will ignore these extra spaces. You should use white space to organize your program code and make it easier to read.

For every opening curly brace (`{`) in a Java program, there must be a corresponding closing curly brace (`}`). The placement of the opening and closing curly braces is not important to the compiler.

Usually, if you vertically align each pair of opening and closing curly braces in your code, it will be easier to read. The following code uses white space and vertically aligns the curly braces:

```
public static void main(String[] args)
{
    System.out.println("First Java program");
}
```

### The `main()` method

The method header for the `main()` method is quite complex. In the method header `public static void main(String[] args)`, the word `public` is an access modifier, just as it is when you defined the `First` class. In the English language, the word `static` means showing little change, or stationary. In the Java programming language, the reserved keyword `static` makes sure that `main()` is accessible even though no objects of the class exist. It also indicates that every member created for the `First` class will have an identical, unchanging `main()` method. Within the Java programming language, `static` also implies uniqueness. Only one `main()` method for the `First` class will ever be stored in the memory of the computer. Of course, other classes eventually might have their own, different `main()` methods.

In English, the word `void` means empty. When the keyword `void` is used in the `main()` method header, it does not indicate that the `main()` method is empty, but rather, that the `main()` method does not return any value when it is called. This does not mean that `main()` does not produce output—in fact, the method does. The `main()` method does not send any value back to any other method that might use it.

Not all classes have a `main()` method; in fact many do not. However, all Java applications must include a method named `main()`, and most Java applications have additional methods. When you execute a Java application, the compiler always executes the `main()` method first.

In the method header `public static void main(String[] args)`, you already might recognize that the contents between the parentheses, `(String[] args)`, must represent an argument passed to the `main()` method, just as the string “First Java program” is an argument passed to the `println()` method. `String` represents a Java class that can be used to represent character strings. The identifier `args` is used to hold any `String` values that might be sent to the `main()` method. The `main()` method could do something with those arguments, such as print them. Nevertheless, you must place an identifier within the `main()` method’s parentheses. The identifier does not need to be named `args`—it could be any legal Java identifier—but the name `args` is traditional.

**Note:** When you refer to the `String` class in the `main()` method header, the square brackets indicate an array of `String` objects.



## Your first program

Now that you understand the basic framework of a program written in Java, you are ready to enter your first Java program into a text editor. It is a tradition among programmers that the first program you write in any language produces “Hello, world!” as its output. To create such a program, you can use any text editor, such as Notepad, Text pad, or any other text-processing program.

**Note:** Many text editors attach their own file name extension (such as .txt or .doc) to a saved file. Double-check your saved file to ensure that it does not have a double extension (such as Hello.java.txt). If the file has a double extension, rename the file. If you explicitly type quotation marks surrounding a file name (such as “Hello.java”), most text editors will save the file as you specify, without adding an extension. Make sure that you save your programs as .java files. The default for Notepad is to save all documents as text.

```
public class Hello
{
    public static void main(String[] args)
    {
    }
}
```

*Exhibit 1-6: The main() method shell for the Hello class*

Do it!

**D-1: Writing a Java program**

Here's how	Here's why
1 Click <b>Start</b>	To start Notepad.
Choose <b>Programs, Accessories, Notepad</b>	
2 Choose <b>File, Save As...</b>	To open the Save As dialog box.
Navigate to the current unit folder	
Save the program as <b>"Hello.java"</b>	(In the current unit folder.) Use quotes around the title to make sure Notepad saves the file with the .java extension instead of a .txt extension. It is important that the file extension is .java. If it is not, the compiler for Java will not recognize the program.
3 Type <b>public class Hello</b>	To specify the class name as Hello. Keep in mind that Java is case sensitive, so a class named Hello is not the same as a class named hello.
4 Press 	You will add the main() method between the curly braces. It is good practice to place each curly brace on its own line. This makes the code easier to read.
Type <b>{</b>	To begin adding the class information.
Press 	
5 Enter the following code:	
<pre>public static void main(String[] args) {</pre>	To add the main() method header and the opening curly brace as shown in Exhibit 1-6.
6 Enter the following code:	
<pre>System.out.println("Hello, world");</pre>	To produce the output, "Hello, world".
7 Enter <b>}</b>	To add the main() method closing curly brace.
8 Enter <b>}</b>	To add the class closing curly brace.
9 Save and close the program	The complete program is shown in Exhibit 1-7.

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

*Exhibit 1-7: Completed Hello program*

## Adding comments

### Explanation



As you can see, even a simple Java program takes several lines of code. Large programs that perform many tasks include even more code, and as you write longer programs, it becomes increasingly difficult to remember why you included steps, or how you intended to use particular variables.

*Program comments* are non-executing statements that you add to a program for the purpose of documentation. Programmers use comments to leave notes for themselves and for others who might read their programs in the future. At the very least, your programs should include comments indicating the program's author, the date, and the program's name or function. The best practice dictates that you should include a brief comment to tell the purpose for each class and its methods.

You can ask students to include additional comments in their code.

As you work through this course, you should add comments as the first three lines of every program. The comments should contain the program name, your name, and the date.

Comments can also serve a useful purpose when you are developing a program. If a program is not performing as expected, you can comment out various statements and subsequently run the program to observe the effect. When you comment out a statement, the compiler will not turn that statement into executable bytecode. When the program runs, the commented out statement does not execute. This helps you pinpoint the location of errant statements in malfunctioning programs.

There are three types of comments in Java:

- *Line comments* start with two forward slashes (//) and continue to the end of the current line. Line comments can appear on a line by themselves or at the end of a line following executable code.
- *Block comments* start with a forward slash and an asterisk (/\*) and end with an asterisk and a forward slash (\*). Block comments can appear on a line by themselves, on a line before executable code, or after executable code. Block comments also can extend across as many lines as needed.
- *Javadoc comments* are a special case of block comments. They begin with a forward slash and two asterisks (/\*\*) and end with an asterisk and a forward slash (\*). You can use javadoc comments to generate documentation with a program named javadoc.

Tell students that the forward slash (/) and the backslash (\) characters are often confused, but they are two distinct characters. Tell them they cannot use them interchangeably.

The Java Development Kit (JDK) includes the javadoc tool that contains classes that you can use when writing programs in the Java programming language. The JDK can be downloaded from the Sun website at <http://java.sun.com/j2se/>.

Exhibit 1-8 shows how comments are used in code.



```
//Demonstrating comments
/* This shows
   that these comments
   don't matter */
System.out.println("Hello");//This line executes
   // up to where the comment started
/**Everything but the println() line
   is a comment. */
```



Exhibit 1-8: Using comments in a program

Do it!

## D-2: Adding comments to a program


Make sure students save the program in the current unit folder.

### Here's how

- 1 Open your Hello.java program
- 2 Save the program as **"Hello2.java"**
- 3 Place the insertion point at the beginning of the file
- 4 Press 
- 5 Press 
- 6 Enter the following line comments:
 

```
// File name Hello2.java
// Written by <your name>
// Written on <today's date>
```
- 7 Scroll to the end of the fourth line in the program
 

Rename the class **Hello2**

Press 

### Here's why

To add comments to the program.

(In the current unit folder.) To create a new class file. You will change the class name Hello to Hello2.

(Before "public class Hello".) You will add comments in this file.

To insert a new line.

To go to the line you added in the previous step.

You should replace the placeholders and include your name and today's date in your comments. Also, make sure you press Enter after typing each comment line.

This line reads "public class Hello".

8 Enter the following block comment:

```
/* This program demonstrates the use of the println() method to print the message Hello, world 2! */
```

9 Edit the output line so it will print **Hello, world 2!**

The output line currently reads "System.out.println("Hello, world");".

10 Update and close the program

To save the completed program. The complete program is shown in Exhibit 1-9.

```
// Filename Hello2.java
// Written by <your name>
// Written on <today's date>
public class Hello2
/* This program demonstrates the use of the println() method to print the message Hello, world 2! */
{
    public static void main(String[ ] args)
    {
        System.out.println("Hello, world 2");
    }
}
```

*Exhibit 1-9: Completed Hello2 program*

## Running a program

Explanation



After you write and save your program, there are two steps that must occur before you can view the program output:

- 1 You must compile the program you wrote (the source code) into bytecode.
- 2 You must use the Java interpreter to translate the bytecode into executable statements.

To compile your source code from the command line, type `javac` followed by the name of the file that contains the source code. For example, to compile a file named `First.java`, you would type `javac First.java` and then press Enter. There will be one of three outcomes:

- You receive a message such as “Bad command or file name.”
- You receive one or more program language error messages.
- You receive no message, which means that the program compiled successfully.

If you receive a message such as “Bad command or file name,” it might mean one of the following:

- You misspelled the command `javac`.
- You misspelled the file name.
- You are not within the correct subfolder or subdirectory on your command line.
- The Java programming language was not installed properly.

If you receive a programming language error message, then there are one or more syntax errors in the source code. A *syntax error* occurs when you introduce typing errors into your program. For example, if your class name is “first” (with a lowercase f) in the source code, but you saved the file as `First.java`, you will get an error message, such as `public class first should not be defined in First.java`, after compiling the program because “first” and “First” are not the same in a case-sensitive language. If this error occurs, you must reopen the text file that contains the source code, make the necessary corrections, and re-compile.

If you receive no error messages after compiling the code in a file named `First.java`, then the program compiled successfully, and a file named `First.class` was created and saved in the same folder as the program text file. After a successful compile, you can run the class file on any computer that has a Java language interpreter.

To run the program from the command line, you type `java First`. Exhibit 1-10 shows the program’s output.

```
Command Prompt
C:\Student Data\Unit_01>java First
This is my First Java Program
C:\Student Data\Unit_01>
```

Exhibit 1-10: Output of the First program

If you receive the following error instead of the program running, then Java cannot find your bytecode file named `First.class`:

```
Exception in thread "main" java.lang.NoClassDefFoundError:▶  
    First
```

One of the places Java tries to find your bytecode file is your current directory. If your bytecode file is in `C:\Student Data\Unit_01`, you should change your current directory to that. To change your directory, type the following command at the prompt and press Enter:

```
cd C:\Student Data\Unit_01
```

The prompt should change to `C:\Student Data\Unit_01>`. If you enter `dir` at the prompt, you should see your `.java` and `.class` files. Now enter `java First` again.



**Note:** If you still have problems, you might have to change your `CLASSPATH` variable. To see if this is necessary, try “prodding” the `CLASSPATH` with the following command:

```
set CLASSPATH=
```

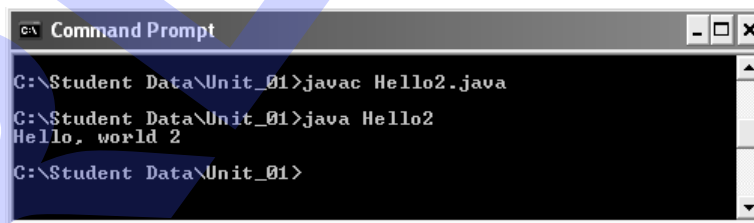
Now enter `java First` again. If the program works now, you will have to change your `CLASSPATH` variable. You set the `CLASSPATH` variable the same way you previously set the `PATH` variable.



*Do it!***D-3: Compiling and interpreting a program**

Here's how	Here's why
<p>1 Click <b>Start</b></p> <p>Choose <b>Programs, Accessories, Command Prompt</b></p>	To open a command prompt.
<p>2 At the command line, enter the following command:</p> <pre>cd C:\Student Data\Unit_01</pre>	To change the directory to the current unit folder.
<p>3 Type <b>javac Hello2.java</b></p> <p>Press </p>	To compile Hello2.java.
<p>4 Type <b>java Hello2</b></p>	To execute the program at the command line.
<p>5 Press </p>	To view the output, as shown in Exhibit 1-11.

*If students receive an error message, help them find and correct the problem.*



```
c:\ Command Prompt
C:\Student Data\Unit_01>javac Hello2.java
C:\Student Data\Unit_01>java Hello2
Hello, world 2
C:\Student Data\Unit_01>
```

*Exhibit 1-11: Output of the Hello2 program*

## Modifying a program

Explanation



After viewing the program output, you might decide to modify the program to get a different result. For example, you might decide to change the First program's output to the following:

```
My new and improved
Java program
```

You can use the following steps to produce the new output:

- 1 Open the First.java file and save it as First2.java.
- 2 Change the class name from First to First2.
- 3 Change the first output statement to:

```
System.out.println("My new and improved");
```
- 4 After the first output statement, add the following output statement:

```
System.out.println("Java program");
```

The completed program is shown in Exhibit 1-12.

```
public class First2
{
    public static void main(String[] args)
    {
        System.out.println("My new and improved");
        System.out.println("Java program");
    }
}
```

Exhibit 1-12: Changing a program's output

However, if you type `java First2` at the command line right now, you will not see the new output—instead, you will see the old output. Before the new source code will execute, you must do the following:

- 1 Compile the First2.java file with the `javac` command.
- 2 Interpret the First2.class bytecode with the `java` command.




## Debugging a program


If you introduce typing errors into your code, the compiler will produce an error message when you compile the code. The exact error message depends on the compiler. In the First program, typing the `System.out.println()` code as `system.out.println("First Java Program");` produces an error message similar to “cannot resolve symbol...” This is a compile-time error commonly referred to as a syntax error. The compiler detects the violation of language rules and refuses to translate the program to bytecode. The compiler reports the errors it finds during compilation so that you can fix the errors and then re-compile the program. Sometimes one error in syntax causes subsequent errors that would not exist if the first syntax error did not exist. You should correct the errors in the order listed by the compiler, and then re-compile the program.

A second kind of error occurs when the syntax of the program is correct but the compiled program produces incorrect results. This is a run-time error or logic error. In the First program, typing the `System.out.println()` code as `System.out.println("Frst Java Program");` does not produce an error. The compiler does not find the spelling error of “Frst” instead of “First”. Errors of this type must be detected by carefully examining the program output. It is the responsibility of the program author to test programs and find any logic errors. Good programming practice stresses programming structure and development that helps minimize errors.

Do it!

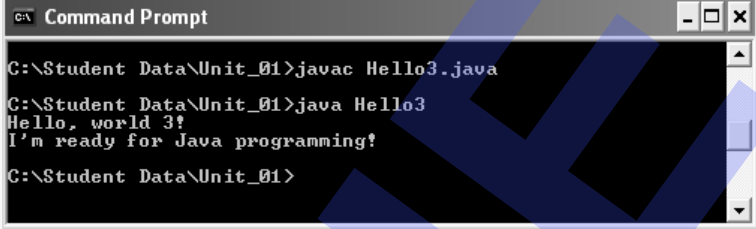
### D-4: Modifying and rerunning a program

Here's how	Here's why
1 Open Hello2.java	You will modify the Hello2 class.
2 Save the program as <b>"Hello3.java"</b>	To create a new class file. You will change the class name Hello2 to Hello3.
3 Edit the first comment line to read <b>// File name Hello3.java</b>	
4 Rename the class <b>Hello3</b>	Edit the line that reads "public class Hello2."
5 Edit the statement that prints Hello, world 2! so that it will print Hello, world 3!	The line currently reads "System.out.println("Hello, world 2");".
6 Place your insertion point after the following statement:  <pre>System.out.println("Hello, world 2");</pre> Press 	
7 Enter the following code:  <pre>System.out.println("I'm ready for Java programming!");</pre>	To add a line that prints "I'm ready for Java programming!"
8 Update and close the program	To save your completed Hello3 program. The complete program is shown in Exhibit 1-13.
9 Switch to the command prompt	
10 Enter <b>javac Hello3.java</b>	To compile the file.  If you receive compile errors, return to the Hello3.java file in the text editor, fix the errors, and then repeat Steps 3 and 4 until the program compiles successfully.
11 Enter <b>java Hello3</b>	To interpret and execute the class. The output should appear as shown in Exhibit 1-14.

 Tell students to make sure they type the semicolon at the end of the statement and use the correct case.

```
// Filename Hello3.java
// Written by <your name>
// Written on <today's date>
public class Hello3
/* This program demonstrates the use of the println() method to print the message Hello, world 3! */
{
    public static void main(String[ ] args)
    {
        System.out.println("Hello, world 3!");
        System.out.println("I'm ready for Java programming!");
    }
}
```

*Exhibit 1-13: Completed Hello3 program*



```
cmd Command Prompt
C:\Student Data\Unit_01>javac Hello3.java
C:\Student Data\Unit_01>java Hello3
Hello, world 3!
I'm ready for Java programming!
C:\Student Data\Unit_01>
```

*Exhibit 1-14: Output of the Hello3 program*

## Unit summary: Getting started

- Topic A** In this topic, you learned programming concepts and object-oriented programming concepts. You also learned how objects are made up of **states** and **methods**. You learned that the states of an object are known as its **attributes**, and an individual **object** is an instance of a **class**, and an object **inherits** its attributes from a class. You also learned that the user of an object does not need to understand the details of any method, but must understand the **interface** with the object.
- Topic B** In this topic, you learned that a program written in Java is run on a standardized hypothetical computer called the **Java Virtual Machine (JVM)**. You also learned that when your program is compiled into **bytecode**, an **interpreter** within the JVM subsequently interprets the bytecode. You also learned about **Java applets** and **Java applications**.
- Topic C** In this topic, you learned how to install the **Java 2 platform**. You also learned the structure of the installed **directory tree**. You learned how to configure the **PATH variable** to conveniently run the **Java 2 SDK executables** such as **javac.exe**, **java.exe**, and **javadoc.exe** from any directory without having to type the full path of the command.
- Topic D** In this topic, you learned how to write a **simple Java program**, which includes a **main () method**. You also learned how to include **comments** in your program code and use comments for **troubleshooting** your programs. Then, you learned how to modify a program. You also learned how to **compile** and **run** a Java program.

### Review questions

- 1 An object's attributes also are known as its \_\_\_\_\_.
  - A states
  - B orientations
  - C methods
  - D procedures
- 2 An instance of a(n) \_\_\_\_\_ inherits its attributes from it.
  - A object
  - B procedure
  - C method
  - D class
- 3 You must compile programs written in Java into \_\_\_\_\_.
  - A bytecode
  - B source code
  - C javadoc statements
  - D object code

- 4 All Java programming language statements must end with a \_\_\_\_\_.
- A period
  - B comma
  - C semicolon**
  - D closing parenthesis
- 5 Arguments to methods always appear within \_\_\_\_\_.
- A parentheses**
  - B double quotation marks
  - C single quotation marks
  - D curly braces
- 6 In a Java program, you must use \_\_\_\_\_ to separate classes, objects, and methods.
- A commas
  - B semicolons
  - C periods**
  - D forward slashes
- 7 All Java programs must have a method named \_\_\_\_\_.
- A `method()`
  - B `main()`**
  - C `java()`
  - D `Hello()`
- 8 Non-executing program statements that provide documentation are called \_\_\_\_\_.
- A classes
  - B notes
  - C comments**
  - D commands
- 9 The Java programming language supports three types of comments: \_\_\_\_\_, \_\_\_\_\_, and javadoc.
- A line, block**
  - B string, literal
  - C constant, variable
  - D single, multiple

- 10 After you write and save a program file, you \_\_\_\_\_ it.
- A interpret and then compile
  - B interpret and then execute
  - C compile and then resave
  - D compile and then interpret**
- 11 The command to execute a compiled program is \_\_\_\_\_.
- A run
  - B execute
  - C javac
  - D java**
- 12 You save text files containing Java language source code using the file extension \_\_\_\_\_.
- A .java**
  - B .class
  - C .txt
  - D .src
- 13 The Java Virtual Machine, or JVM, refers to a(n) \_\_\_\_\_.
- A interpreter**
  - B operating system
  - C real computer
  - D compiler

### Independent practice activity

Sample solutions for the following activities are provided in the current unit's solutions folder.

- 1 Write, compile, and test a program that prints your first name on the screen. Save the program as Name.java in the current unit folder.
- 2 Write, compile, and test a program that prints your full name, street address, city, state, and zip code on three separate lines on the screen. Save the program as Address.java in the current unit folder.



# Unit 2

## Using data in a program

**Unit time: 120 minutes**

Complete this unit, and you will know how to:

- A** Use constants and variables.
- B** Use various data types.

## Topic A: Constants and variables

Explanation



Data can be categorized as being variable or constant. Data is *constant* when it can't be changed after a program is compiled; data is *variable* when it might change. For example, if you include the statement `System.out.println(459);` the number 459 is a constant. Every time the program containing the constant 459 is executed, the value 459 will print. You can refer to the number 459 as a *literal constant* because its value is taken literally at each use.

You can also set your data as a variable. For example, if you create a variable named `ovenTemperature` and include `System.out.println(ovenTemperature);` as a statement, then different values might display each time the program is executed, depending on what value is stored in the `ovenTemperature` variable each time the program is run.

Do it!

### A-1: Discussing constants and variables

#### Questions and answers

- When data can't be changed after a program is compiled, the data is \_\_\_\_\_.
  - constant
  - variable
  - volatile
  - mutable
- What do you use in your program when you have data that might change?

**variables**

### Variable declaration

Explanation



Variables are named memory locations that your program can use to store values. You name variables by using the same naming rules as for legal class identifiers. Variable names must start with a letter and can't be any of the reserved keywords. You must declare all variables you want to use in a program.

A variable declaration includes the following:

- A data type that identifies the type of data that the variable will store
- An identifier that is the variable's name
- An optional assigned value, when you want a variable to contain an initial value
- An ending semicolon

**Note:** Variable names usually begin with lowercase letters to distinguish them from class names, however, variable names can begin with either an uppercase or a lowercase letter.

### Declaring a variable without an initial value

You can declare variables, which are not assigned a value at the time of creation. For example, here is a statement that declares a variable that can store integers:

```
int ovenTemperature;
```

The data type is `int`, the variable name is `ovenTemperature`, and the semi-colon marks the end of the statement. The variable, `ovenTemperature`, can only store values of type `int`.

### Declaring a variable with an initial value

You can also declare a variable and assign a value at the time of creation. For example:

```
int myAge = 25;
```

The data type is `int`, the variable name is `myAge`, the initial value is `25`, and the semi-colon marks the end of the statement. The equals sign (`=`) is the *assignment operator*. Any value to the right of the equals sign is assigned to the variable on the left of the equals sign. An assignment made when you declare a variable is an *initialization*; an assignment made later is simply an *assignment*. Thus, `int myAge = 25;` initializes `myAge` to `25`. A subsequent statement such as `myAge = 42;` might assign a new value to the variable. You should note that the expression `25 = myAge;` is illegal.

### Declaring multiple variables

You can declare multiple variables of the same type in separate statements on different lines. For example, the following statements declare two variables—the first variable is named `myAge` and its value is `25`. The second variable is named `yourAge` and its value is `19`.

```
int myAge = 25;  
int yourAge = 19;
```

You can also declare two variables of the same type in a single statement by separating the variable declarations with a comma, shown in the following statement:


```
int myAge = 25, yourAge = 19;
```


If you want to declare variables of different types, you must use a separate statement for each type. The following statements declare two variables of type `int` (`myAge` and `yourAge`) and two variables of type `double` (`mySalary` and `yourSalary`):

```
int myAge, yourAge;  
double mySalary, yourSalary;
```


Do it!

**A-2: Declaring variables**

 Make sure students save the program to the current unit folder.

Here's how	Here's why
1 Open a new text file	You will declare and display values in a program.
2 Save the program as <b>DemoVariables.java</b>	Save the program in the current unit folder.
3 Enter the following code:  <pre>public class DemoVariables { }</pre>	To write a class header.
4 Place the insertion point after the opening curly brace  Press   Enter the following code:  <pre>public static void main(String[] args) { }</pre>	To enter a new line.  To write the main() method.
5 In the <code>main()</code> method, place the insertion point after the opening curly brace  Press   Enter <b><code>int oneInt = 315;</code></b>	To enter a new line.  To declare a variable of type <code>int</code> named <code>oneInt</code> with a value of 315. You can declare variables at any point within a method prior to their first use, however, it is a common practice to declare variables first, and place method calls second.
6 Enter the following code:  <pre>System.out.print("The int is "); System.out.println(oneInt);</pre>	To write the two output statements. The first statement uses the <code>print()</code> method to output "The int is " and leaves the insertion point on the same output line. The second statement uses the <code>println()</code> method to output the value of <code>oneInt</code> and advances the insertion point to a new line.

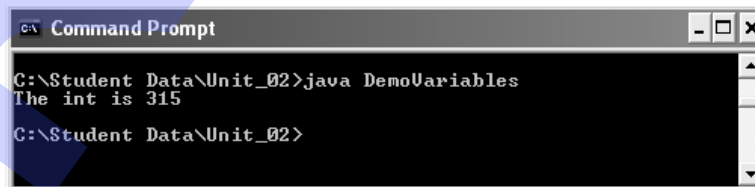
Tell students if they type a space before the closing quotation mark in a literal string such as "The int is ", a space will be added between the end of the literal string and the value that prints.

 Make sure students change the directory to the current unit folder.

7 Update and close the program	The completed program is shown in Exhibit 2-1.
8 Switch to the command prompt	If necessary.
9 At the command prompt, enter:  <code>cd C:\Student Data\Unit_02</code>	To change the directory to the current unit folder.
10 Enter the following command:  <code>javac DemoVariables.java</code>	To compile the file. If you encounter any errors, correct the errors, save the program, and compile it again.
11 Enter the following command:  <code>java DemoVariables</code>	To execute the program.
12 Examine the output	As shown in Exhibit 2-2.

```
public class DemoVariables
{
    public static void main(String[] args)
    {
        int oneInt = 315;
        System.out.print("The int is ");
        System.out.println(oneInt);
    }
}
```

*Exhibit 2-1: Completed DemoVariables program*



```
CA Command Prompt
C:\Student Data\Unit_02>java DemoVariables
The int is 315
C:\Student Data\Unit_02>
```

*Exhibit 2-2: Output of the DemoVariables program*

Do it!

### A-3: Discussing variable declaration

#### Questions and answers

- 1 Which of the following elements is optional in a variable declaration?
  - A A type
  - B An identifier
  - C** An assigned value
  - D A semicolon
- 2 The statement `int i;` is an example of \_\_\_\_\_.
  - A initialization
  - B assignment
  - C** declaration
  - D constant
- 3 To distinguish them from class names, variable names usually begin with \_\_\_\_\_ letters.

***lowercase***

## Topic B: Data types and character sets

### Explanation

Now you will learn how to use primitive data types. You will also learn about ASCII and Unicode character sets.

### Primitive data types



The Java programming language provides the following eight primitive data types:

- boolean
- float
- byte
- int
- char
- long
- double
- short

These eight data types are called *primitive types* because they are simple and uncomplicated. Primitive types serve as the building blocks for more complex data types, called *reference types*.

### The int data type



You use variables of type `int` to store (or hold) integers, or whole numbers. An integer can hold any whole number value from  $-2,147,483,648$  to  $2,147,483,647$ .

When assigning a value to an `int` variable, you do not type any commas; you type only digits and an optional plus or minus sign to indicate a positive or negative integer.

**Note:** The legal integer values are  $-2^{31}$  through  $2^{31}-1$ . These are the highest and lowest values that you can store in four bytes of memory, which is the size of an `int` variable.

The types `byte`, `short`, and `long` are all variations of the integer type. You use `byte` or `short` if you know a variable will need to hold only small values so you can save space in memory. Use a `long` type if you know you will be working with large values. The range for each of these types are shown in the following table:



Type	Range	Size in bytes
<code>byte</code>	$-128$ to $127$	1
<code>short</code>	$-32,768$ to $32,767$	2
<code>int</code>	$-2,147,483,648$ to $2,147,483,647$	4
<code>long</code>	$-9,223,372,036,854,775,808$ to $9,223,372,036,854,775,807$	8

It is important to choose appropriate types for the variables you will use in a program. If you attempt to assign a value that is too large for the data type of the variable, the compiler will issue an error message and the program will not execute. If you choose a data type that is larger than you need, you waste memory. For example, a personnel program might use a `byte` variable for number of dependents (a limit of 127 is more than enough), a `short` variable for hours worked in a month (127 is not enough), and an `int` variable for an annual salary. Even though a limit of 32,000 might be large enough for your salary, it is not enough for the CEO.

If your program uses a literal constant integer, such as 932, the integer is an `int` type by default. If you need to use a constant higher than 2,147,483,647, you must follow the number with the letter `L` to indicate `long` as shown in the following statement:



```
long mosquitosInTheNorthWoods = 2444555888L;
```

The statement stores a number that is greater than the maximum limit for the `int` type. You can type either an uppercase or lowercase `L` to indicate the `long` type, but the uppercase `L` is preferred to avoid confusion with the number one.



Do it!

**B-1: Using int data type**

Here's how	Here's why
1 Open DemoVariables.java	You will declare variables of int data type.
2 Save the program as <b>DemoVariables2.java</b>	
3 Rename the class DemoVariables to <b>DemoVariables2</b>	
4 Place the insertion point after <b>int oneInt = 315;</b>  Press 	To start a new line.
Enter the following code:  <pre>short oneShort = 23; long oneLong = 1234567876543L;</pre>	
5 Place the insertion point after the following statement:  <pre>System.out.println(oneInt);</pre> Press 	
Enter the following code:  <pre>System.out.print("The short is "); System.out.println(oneShort); System.out.print("The long is "); System.out.println(oneLong);</pre>	To display the values of the two new variables.
6 Update the program	The completed program is shown in Exhibit 2-3.
7 Switch to the command prompt	
8 Compile the program	At the command prompt, enter <code>javac DemoVariables2.java</code> .
9 Execute the program	At the command prompt, enter <code>java DemoVariables2</code> .
10 Examine the output	As shown in Exhibit 2-5.
11 Switch to DemoVariables2.java	
12 Save the program as <b>DemoVariables3.java</b>	

Tell students they can also enter this code from "B1Code1.txt" file from the current unit folder.

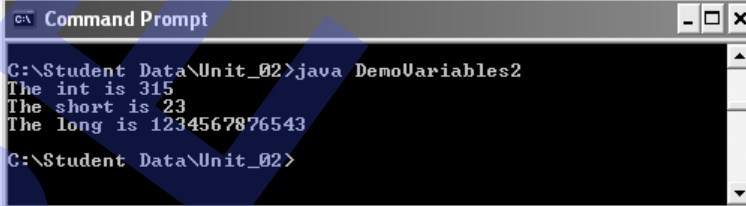
<p>13 Rename the class DemoVariables2 to <b>DemoVariables3</b></p>	<p>To write a program where the two print methods are combined into a single statement.</p>
<p>14 Select the following statements:</p> <pre>System.out.print("The int is "); System.out.println(oneInt);</pre> <p>Press <input type="button" value="DELETE"/></p> <p>In place of the deleted statements, type:</p> <pre>System.out.println("The int is " + oneInt);</pre>	<p>To delete the selected text.</p>
<p>15 Select the following statements:</p> <pre>System.out.print("The short is "); System.out.println(oneShort);</pre> <p>Press <input type="button" value="DELETE"/></p> <p>In place of the deleted statements, type:</p> <pre>System.out.println("The short is " + oneShort);</pre>	<p>To delete the two selected statements.</p>
<p>16 Delete the following statements:</p> <pre>System.out.print("The long is "); System.out.println(oneLong);</pre> <p>In place of the deleted statements, type:</p> <pre>System.out.println("The long is " + oneLong);</pre>	
<p>17 Update and close the program</p>	<p>The completed program is shown in Exhibit 2-4.</p>
<p>18 Compile and run the program</p>	<p>Notice the output for this program is the same as the output for the DemoVariables2 program, as shown in Exhibit 2-5.</p>

```
public class DemoVariables2
{
    public static void main(String[] args)
    {
        int oneInt = 315;
        short oneShort = 23;
        long oneLong = 1234567876543L;
        System.out.print("The int is ");
        System.out.println(oneInt);
        System.out.print("The short is ");
        System.out.println(oneShort);
        System.out.print("The long is ");
        System.out.println(oneLong);
    }
}
```

*Exhibit 2-3: Completed DemoVariables2 program*

```
public class DemoVariables3
{
    public static void main(String[] args)
    {
        int oneInt = 315;
        short oneShort = 23;
        long oneLong = 1234567876543L;
        System.out.println("The int is " + oneInt);
        System.out.println("The short is " + oneShort);
        System.out.println("The long is " + oneLong);
    }
}
```

*Exhibit 2-4: Completed DemoVariables3 program*



```
CA Command Prompt
C:\Student Data\Unit_02>java DemoVariables2
The int is 315
The short is 23
The long is 1234567876543
C:\Student Data\Unit_02>
```

*Exhibit 2-5: Output of the DemoVariables2 program*

## Arithmetic operators

Explanation

Use the arithmetic operators to manipulate values in your programs. The following table describes five standard arithmetic operators for integers:



Operator	Description	Example
+	Addition	45 + 2, the result is 47
-	Subtraction	45 - 2, the result is 43
*	Multiplication	45 * 2, the result is 90
/	Division	45 / 2, the result is 22 (not 22.5)
%	Modulus (remainder)	45 % 2, the result is 1 (that is, 45 / 2 = 22 with a remainder of 1)

**Note:** You do not need to perform a division operation before you perform a modulus operation. A modulus operation can stand-alone.

When you divide two integers, whether they are integer constants or integer variables, the result is an integer, any fractional part of the result is lost. For example, the result of 45 / 2 is 22, even though the result is 22.5 in a mathematical expression. When you use the modulus operator with two integers, the result is an integer with the value of the remainder after division takes place—the result of 45 % 2 is 1 because 2 “goes into” 45 twenty-two times with a remainder of 1.



When you combine mathematical operations in a single statement, you must understand operator *precedence*, or the order in which parts of a mathematical expression are evaluated. Multiplication, division, and modulus always take place prior to addition or subtraction in an expression. For example, consider the following expression:

```
int result = 2 + 3 * 4;
```

The expression results in 14, because the multiplication (3 \* 4) occurs before adding 2. You can override normal operator precedence by putting the operation to perform first in parentheses. Consider the following expression with parentheses:

```
int result = (2 + 3) * 4;
```

The expression results in 20, because the addition within the parentheses takes place first, and then that result (5) is multiplied by 4.

Do it!

### B-2: Using arithmetic operators

Here's how	Here's why
1 Open DemoVariables3.java	You will write a program that uses arithmetic statements.
2 Save the program as <b>DemoVariables4.java</b>	
3 Rename the class DemoVariables3 to <b>DemoVariables4</b>	

- 4 Place the insertion point after the following statement:

```
long oneLong = 1234567876543L;
```

Press **↵ ENTER**

To start a new line.

Type the following code:

```
int value1 = 43, value2 = 10, sum, difference, product,
    quotient, modulus;
```

- 5 Place the insertion point after the following statement:

```
System.out.println("The long is " + oneLong);
```

Press **↵ ENTER**

Enter the following code:

```
sum = value1 + value2;
difference = value1 - value2;
product = value1 * value2;
quotient = value1 / value2;
modulus = value1 % value2;
```

To write the code for arithmetic statements.

- 6 Enter the following code:

```
System.out.println("Sum is " + sum);
System.out.println("Difference is " + difference);
System.out.println("Product is " + product);
System.out.println("Quotient is " + quotient);
System.out.println("Modulus is " + modulus);
```

To write the code for output statements.

- 7 Update and close the program

The completed program is shown in Exhibit 2-6.

- 8 Compile and run the program

To execute the output.

Examine the output

To analyze the output and confirm that the arithmetic is correct, as shown in Exhibit 2-7.

*Tell students the arrow indicates that the line wraps and they should not insert a hard return.*

*Tell students they can also enter this code from "B2Code1.txt" file from the current unit folder.*

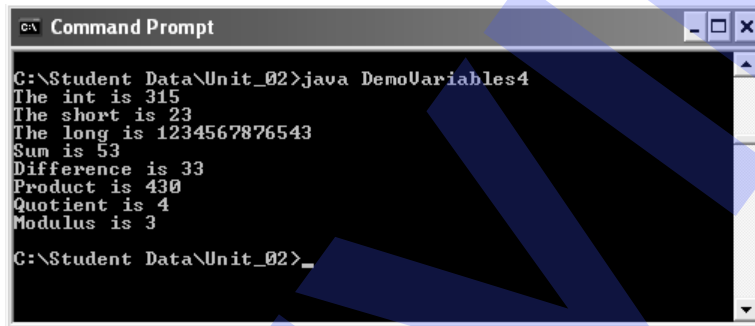
*Tell students they can also enter this code from "B2Code2.txt" file from the current unit folder.*

```

public class DemoVariables4
{
    public static void main(String[] args)
    {
        int oneInt = 315;
        short oneShort = 23;
        long oneLong = 1234567876543L;
        System.out.println("The int is " + oneInt);
        System.out.println("The short is " + oneShort);
        System.out.println("The long is " + oneLong);
        int value1 = 43, value2 = 10, sum, difference, product, quotient, modulus;
        sum = value1 + value2;
        difference = value1 - value2;
        product = value1 * value2;
        quotient = value1 / value2;
        modulus = value1 % value2;
        System.out.println("Sum is " + sum);
        System.out.println("Difference is " + difference);
        System.out.println("Product is " + product);
        System.out.println("Quotient is " + quotient);
        System.out.println("Modulus is " + modulus);
    }
}

```

Exhibit 2-6: Completed DemoVariables4 program



```

C:\Student Data\Unit_02>java DemoVariables4
The int is 315
The short is 23
The long is 1234567876543
Sum is 53
Difference is 33
Product is 430
Quotient is 4
Modulus is 3
C:\Student Data\Unit_02>_

```

Exhibit 2-7: Output of the DemoVariables4 program

Do it!

**B-3: Discussing arithmetic operators****Questions and answers**

- The modulus operator \_\_\_\_\_.
  - is represented by a forward slash
  - provides the remainder of integer division
  - provides the quotient of integer division
  - Both B and C
- According to the rules of operator precedence, division always takes place prior to \_\_\_\_\_.
  - multiplication
  - modulus
  - subtraction
  - Both A and B

**The boolean data type***Explanation*

Boolean logic is based on true-or-false comparisons. A boolean variable can hold only one of two values—true or false. You can declare and assign appropriate values to boolean variables as:

```
boolean isItPayday = false;
boolean areYouBroke = true;
```

You can assign values based on the result of comparisons to boolean variables. A *comparison operator* compares items; an expression containing a comparison operator has a boolean value. The following table describes Java's six comparison operators:



<b>Operator</b>	<b>Example (true)</b>	<b>Example (false)</b>
< (Less than)	3 < 8	8 < 3
> (Greater than)	4 > 2	2 > 4
== (Equal to)	7 == 7	3 == 9
<= (Less than or equal to)	5 <= 5	8 <= 6
>= (Greater than or equal to)	7 >= 3	1 >= 2
!= (Not equal to)	5 != 6	3 != 3

When you use any of the operators that have two symbols (==, <=, >=, or !=), you cannot place any white space between the two symbols, however, you can place white space around the symbols to increase legibility.

Legal, declaration statements might include the following statements, which compare two values directly:

```
boolean isSixBigger = (6 > 5);
// Value stored would be true
boolean isSevenSmallerOrEqual = (7 <= 4);
// Value stored would be false
```

**Note:** Variable names are easily identified as boolean if you use a form of “to be” (such as “is” or “are”) as part of the variable name.



The boolean expressions are more meaningful when variables that have been assigned values, are used in the comparisons, as in the following examples:

```
boolean overtime = (hours > 40);
boolean highTaxBracket = (income > 100000);
```

In the first statement, the `hours` variable is compared to a constant value of 40. If the `hours` variable is not greater than 40, then the expression evaluates to false. In the second statement, the `income` variable must be greater than 100000 for the expression to evaluate to true.

Do it!

**B-4: Adding boolean variables**

Here's how	Here's why
1 Open DemoVariables4.java	
2 Save the program as <b>DemoVariables5.java</b>	
3 Rename the class DemoVariables4 to <b>DemoVariables5</b>	
4 Place the insertion point after <b>int oneInt = 315;</b>	
Press 	To insert a new line.
Type the following code:	
<pre>boolean isProgrammingFun = true, isProgrammingHard = false;</pre>	To declare two new boolean variables in the program.
5 Press 	
Type the following code:	
<pre>System.out.println("The value of isProgrammingFun is " +     isProgrammingFun); System.out.println("The value of isProgrammingHard is " +     isProgrammingHard);</pre>	To add print statements that will display values.
6 Update and close the program	The completed program is shown in Exhibit 2-8.
7 Compile and run the program	
Examine the output	As shown in Exhibit 2-9.

*Remind students that the code shown here wraps to the next line and they should not press Enter.*

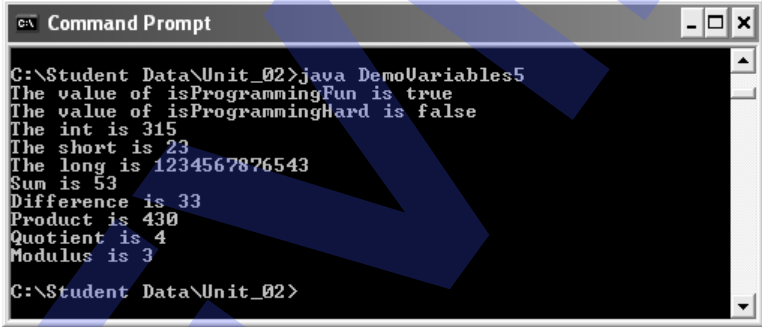


```
public class DemoVariables5
{
    public static void main(String[] args)
    {
        int oneInt = 315;
        boolean isProgrammingFun = true, isProgrammingHard = false;
        System.out.println("The value of isProgrammingFun is " + isProgrammingFun);
        System.out.println("The value of isProgrammingHard is " + isProgrammingHard);

        short oneShort = 23;
        long oneLong = 1234567876543L;
        System.out.println("The int is " + oneInt);
        System.out.println("The short is " + oneShort);
        System.out.println("The long is " + oneLong);

        int value1 = 43, value2 = 10, sum, difference, product, quotient, modulus;
        sum = value1 + value2;
        difference = value1 - value2;
        product = value1 * value2;
        quotient = value1 / value2;
        modulus = value1 % value2;
        System.out.println("Sum is " + sum);
        System.out.println("Difference is " + difference);
        System.out.println("Product is " + product);
        System.out.println("Quotient is " + quotient);
        System.out.println("Modulus is " + modulus);
    }
}
```

*Exhibit 2-8: Completed DemoVariables5 program*



```
Command Prompt
C:\Student Data\Unit_02>java DemoVariables5
The value of isProgrammingFun is true
The value of isProgrammingHard is false
The int is 315
The short is 23
The long is 1234567876543
Sum is 53
Difference is 33
Product is 430
Quotient is 4
Modulus is 3
C:\Student Data\Unit_02>
```

*Exhibit 2-9: Output of the DemoVariables5 program*

## Floating-point data types

Explanation



A *floating-point* number contains decimal positions. Java supports two floating-point data types: `float` and `double`. A `float` data type can hold values up to six or seven significant digits of accuracy. A `double` data type can hold 14 or 15 significant digits of accuracy. The term *significant digits* refer to the mathematical accuracy of a value. For example, a `float` variable given the value 0.324616777 will display as 0.324617 because the value is only accurate to the sixth decimal position. The range for floating-point values are shown in the following table:



Type	Range	Size in bytes
<code>float</code>	$-3.4 * 10^{38}$ to $3.4 * 10^{38}$	4
<code>double</code>	$-1.7 * 10^{308}$ to $1.7 * 10^{308}$	8


Just as an integer constant, such as 178, is an `int` by default, a floating-point number constant such as 18.23 is a `double` by default. To store a value explicitly as a `float`, you can type the letter F after the number, as shown in the following statement:

```
float pocketChange = 4.87F;
```

You can type either a lowercase or an uppercase F, you can also type D (or d) after a floating-point value to indicate it is a `double`. Even without the D, the value will be stored as a `double` by default.

As with `int` types, you can perform the mathematical operations of addition, subtraction, multiplication, and division with floating-point numbers; however, you cannot perform modulus operations using floating-point values. Floating-point division yields a floating-point result, there is no remainder.

*Do it!***B-5: Using floating-point data types**

Here's how	Here's why
1 Open DemoVariables5.java	You will add floating-point variables to a program.
2 Save the program as <b>DemoVariables6.java</b>	
3 Rename the class DemoVariables5 to <b>DemoVariables6</b>	
4 Place the insertion point after the following statement:  <pre>boolean isProgrammingFun = true, isProgrammingHard = false;</pre> Press   Enter the following code:  <pre>double doubNum1 = 2.3, doubNum2 = 14.8, doubResult;</pre>	To start a new line.  To add three new floating-point variables.
5 Enter the following code:  <pre>doubResult = doubNum1 + doubNum2; System.out.println("The sum of the doubles is " +     doubResult); doubResult = doubNum1 * doubNum2; System.out.println("The product of the doubles is "+     doubResult);</pre>	To perform arithmetic and produce output.
6 Update and close the program	The completed program is shown in Exhibit 2-10.
7 Compile and run the program  Examine the output	As shown in Exhibit 2-11.

*Tell students they can also enter this code from the "B4Code1.txt" file from the current unit folder.*

*Remind students that the code shown here wraps to the next line and they should not press Enter.*

```

public class DemoVariables6
{
    public static void main(String[] args)
    {
        int oneInt = 315;
        boolean isProgrammingFun = true, isProgrammingHard = false;
        double doubNum1 = 2.3, doubNum2 = 14.8, doubResult;
        doubResult = doubNum1 + doubNum2;
        System.out.println("The sum of the doubles is " + doubResult);

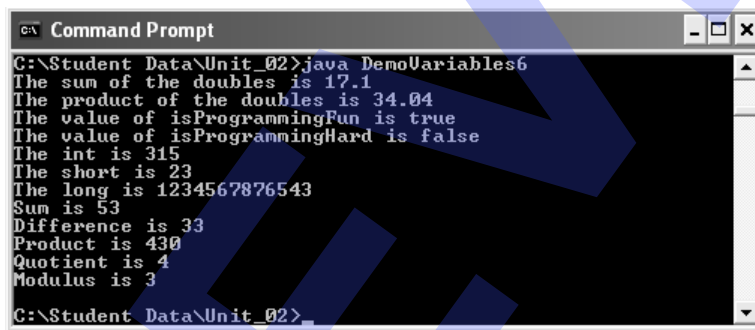
        doubResult = doubNum1 * doubNum2;
        System.out.println("The product of the doubles is " + doubResult);
        System.out.println("The value of isProgrammingFun is " + isProgrammingFun);
        System.out.println("The value of isProgrammingHard is " + isProgrammingHard);

        short oneShort = 23;
        long oneLong = 1234567876543L;
        System.out.println("The int is " + oneInt);
        System.out.println("The short is " + oneShort);
        System.out.println("The long is " + oneLong);

        int value1 = 43, value2 = 10, sum, difference, product, quotient, modulus;
        sum = value1 + value2;
        difference = value1 - value2;
        product = value1 * value2;
        quotient = value1 / value2;
        modulus = value1 % value2;
        System.out.println("Sum is " + sum);
        System.out.println("Difference is " + difference);
        System.out.println("Product is " + product);
        System.out.println("Quotient is " + quotient);
        System.out.println("Modulus is " + modulus);
    }
}

```

Exhibit 2-10: Completed DemoVariables6 program



```

C:\Student Data\Unit_02>java DemoVariables6
The sum of the doubles is 17.1
The product of the doubles is 34.04
The value of isProgrammingFun is true
The value of isProgrammingHard is false
The int is 315
The short is 23
The long is 1234567876543
Sum is 53
Difference is 33
Product is 430
Quotient is 4
Modulus is 3
C:\Student Data\Unit_02>

```

Exhibit 2-11: Output of the DemoVariables6 program

*Do it!***B-6: Discussing floating-point data types****Questions and answers**

- 1 The value 137.68 can be held by a variable of type \_\_\_\_\_.
  - A int
  - B float
  - C double
  - D Both B and C**
  
- 2 A float value is accurate to which decimal position?
  - A Sixth**
  - B Seventh
  - C Eighth
  - D Tenth

## Numeric type conversion

### Explanation

When you are performing arithmetic with variables or constants of the same type, the result of the arithmetic retains the same type. For example, when you divide two `ints`, the result is an `int`, and when you subtract two `doubles`, the result is a `double`. Often, however, you might want to perform mathematical operations on unlike types. In the following example you multiply an `int` by a `double`:

```
int hoursWorked = 37;
double payRate = 6.73;
double grossPay = hoursWorked * payRate;
```

When you perform arithmetic operations with operands of unlike types, Java chooses a unifying type for the result. Java then implicitly (or automatically) converts nonconforming operands to the unifying type.

**Note:** An *operand* is simply any value used in an arithmetic or logical operation.

The following list of operands ranks the order for establishing unifying types between two variables:

- `double`
- `float`
- `long`
- `int`
- `short`
- `byte`

In other words, `grossPay` is the result of multiplication of an `int` and a `double`, so `grossPay` itself must be a `double`. If `grossPay` is the result of the addition of a `short` and an `int`, then the result would be an `int`.

You can explicitly, or purposely, override the unifying type imposed by the Java programming language by performing a type cast. *Type casting* involves placing the desired result type in parentheses, followed by the variable or constant to be cast. For example, two type casts are performed in the following code:

```
double bankBalance = 189.66;
float weeklyBudget = (float) bankBalance / 4;
// weeklyBudget is 47.415, one-fourth of bankBalance
int dollars = (int) weeklyBudget;
// dollars is 47, the integer part of weeklyBudget
```

The `double` value `bankBalance / 4` is converted to a `float` before it is stored in `weeklyBudget`, and the `float` value `weeklyBudget` is converted to an `int` before it is stored in `dollars`. When the `float` value is converted to an `int`, the decimal place values are lost.

You can lose data when performing a cast. For example, the largest `byte` value is 127 and the largest `int` value is 2,147,483,647, the following statements produce distorted results:

```
int anOkayInt = 200;
byte aBadByte = (byte)anOkayInt;
```

A byte is constructed from eight 1s and 0s, or binary digits. The first binary digit, or bit, holds a 0 or 1 to represent positive or negative. The remaining seven bits store the actual value. When the integer value 200 is stored in the `byte` variable, its large value consumes the eighth bit, turning it to a 1, and forcing the `aBadByte` variable to appear to hold the value `-72` that is inaccurate and misleading.

Do it!

### **B-7: Discussing numeric type conversion**

#### **Questions and answers**

- 1 You use a \_\_\_\_\_ to explicitly override an implicit type.  
A mistake  
**B type cast**  
C format  
D type set
- 2 When adding a float, a double, and an int, the result is \_\_\_\_\_.  
A float  
**B double**  
C int  
D long
- 3 Which of the following operations yield 1 as an answer?  
**A 13 % 4**  
B 13.0 % 4.0  
C 8 % 4  
D 8.0 % 4.0
- 4 When you perform mathematical operations on unlike types, Java \_\_\_\_\_ converts the variables to a unifying type.  
**implicitly**
- 5 Converting from one data type to another in Java is done implicitly and \_\_\_\_\_.  
**explicitly**

## The char data type

### Explanation



You use the `char` data type to hold any single character. You place constant character values within single quotation marks because the computer stores characters and integers differently.

For example, the statements `char aCharValue = '9';` and `int aNumValue = 9;` are legal. The statements `char aCharValue = 9;` and `int aNumValue = '9';` might produce undesirable results. Consider the following `println()` statement:

```
System.out.println("aCharValue is " + aCharValue + >
    "aNumValue is " + aNumValue);
```

When these variables are used in the statement, the resulting output is a blank and the number 57, which are ASCII codes. Exhibit 2-12 shows ASCII decimal codes and character equivalents. A number can be a character, but it must be enclosed in single quotation marks and declared as a `char` type, however, you cannot store an alphabetic letter in a numeric type. The following code shows how you can store any character string by using the `char` data type:

```
char myInitial = 'J';
char percentSign = '%';
char numThatIsAChar = '9';
```

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	nul	32	!	64	@	96	`
1	soh ^A	33	"	65	A	97	a
2	stx ^B	34	#	66	B	98	b
3	etx ^C	35	\$	67	C	99	c
4	eot ^D	36	%	68	D	100	d
5	enq ^E	37	&	69	E	101	e
6	ack ^F	38	'	70	F	102	f
7	bel ^G	39	(	71	G	103	g
8	bs ^H	40	)	72	H	104	h
9	ht ^I	41	*	73	I	105	i
10	lf ^J	42	+	74	J	106	j
11	vt ^K	43	,	75	K	107	k
12	ff ^L	44	-	76	L	108	l
13	cr ^M	45	.	77	M	109	m
14	so ^N	46	/	78	N	110	n
15	si ^O	47	0	79	O	111	o
16	dle ^P	48	1	80	P	112	p
17	dc1 ^Q	49	2	81	Q	113	q
18	dc2 ^R	50	3	82	R	114	r
19	dc3 ^S	51	4	83	S	115	s
20	dc4 ^T	52	5	84	T	116	t
21	nak ^U	53	6	85	U	117	u
22	syn ^V	54	7	86	V	118	v
23	etb ^W	55	8	87	W	119	w
24	can ^X	56	9	88	X	120	x
25	em ^Y	57	:	89	Y	121	y
26	sub ^Z	58	;	90	Z	122	z
27	esc	59	<	91	[	123	{
28	fs	60	=	92	\	124	
29	gs	61	>	93	]	125	}
30	rs	62	?	94	^	126	~
31	us	63		95	_	127	del

Exhibit 2-12: ASCII character set





A `char` type variable can hold only one character. To store a string of characters, such as a person's name, you must use a data structure called a *String*. Unlike single characters, that use single quotation marks, string constants are written between double quotation marks. For example, the expression that stores the name Audrey as a string in a variable named `firstName` is `String firstName = "Audrey";`.

You can store any character—including nonprinting characters such as a backspace or a tab—in a `char` variable. To store these characters, you must use an escape sequence, which always begins with a backslash. For example, the following code stores a backspace character and a tab character in the `char` variables `aBackspaceChar` and `aTabChar`:

```
char aBackspaceChar = '\b';
char aTabChar = '\t';
```



In the preceding code, the escape sequence indicates a unique value for the character, instead of the letters `b` or `t`. Some common escape sequences are listed in the following table:



<b>Escape sequence</b>	<b>Description</b>
<code>\b</code>	Backspace
<code>\t</code>	Tab
<code>\n</code>	Newline or linefeed
<code>\f</code>	Form feed
<code>\r</code>	Carriage return
<code>\"</code>	Double quotation mark
<code>'</code>	Single quotation mark
<code>\\</code>	Backslash

Do it!

**B-8: Using escape sequences**

Here's how	Here's why
1 Open DemoVariables6.java	
2 Save the program as <b>DemoVariables7.java</b>	
3 Rename the class DemoVariables6 to <b>DemoVariables7</b>	
4 Place the insertion point before the last curly brace in the program	In the last line of the program.
Press 	To create a new line.
Press 	
Type the following code:	
<pre>System.out.println("\nThis is on one line\nThis on another"); System.out.println("This shows\tchow\ttabs\twork");</pre>	
5 Update and close the program	The completed program is shown in Exhibit 2-13.
6 Compile and run the program	To execute the output.
Examine the output	As shown in Exhibit 2-14.

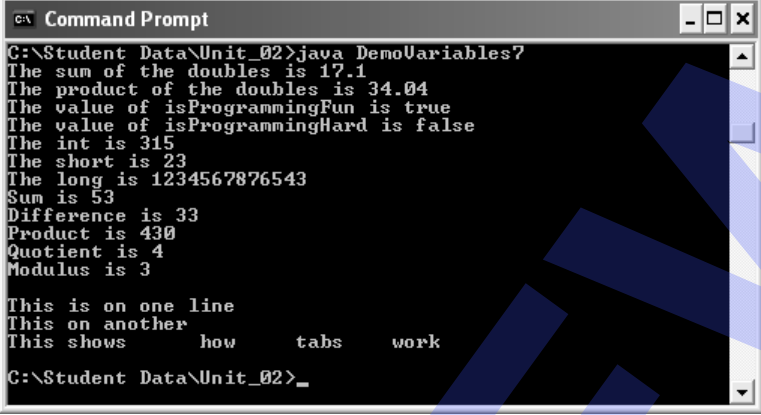
```
public class DemoVariables7
{
    public static void main(String[] args)
    {
        int oneInt = 315;
        boolean isProgrammingFun = true, isProgrammingHard = false;
        double doubNum1 = 2.3, doubNum2 = 14.8, doubResult;
        doubResult = doubNum1 + doubNum2;
        System.out.println("The sum of the doubles is " + doubResult);

        doubResult = doubNum1 * doubNum2;
        System.out.println("The product of the doubles is " + doubResult);
        System.out.println("The value of isProgrammingFun is " + isProgrammingFun);
        System.out.println("The value of isProgrammingHard is " + isProgrammingHard);

        short oneShort = 23;
        long oneLong = 1234567876543L;
        System.out.println("The int is " + oneInt);
        System.out.println("The short is " + oneShort);
        System.out.println("The long is " + oneLong);

        int value1 = 43, value2 = 10, sum, difference, product, quotient, modulus;
        sum = value1 + value2;
        difference = value1 - value2;
        product = value1 * value2;
        quotient = value1 / value2;
        modulus = value1 % value2;
        System.out.println("Sum is " + sum);
        System.out.println("Difference is " + difference);
        System.out.println("Product is " + product);
        System.out.println("Quotient is " + quotient);
        System.out.println("Modulus is " + modulus);
        System.out.println("\nThis is on one line\nThis on another");
        System.out.println("This shows\tchow\ttabs\twork");
    }
}
```

Exhibit 2-13: Completed DemoVariables7 program



```

C:\Student Data\Unit_02>java DemoVariables7
The sum of the doubles is 17.1
The product of the doubles is 34.04
The value of isProgrammingFun is true
The value of isProgrammingHard is false
The int is 315
The short is 23
The long is 1234567876543
Sum is 53
Difference is 33
Product is 430
Quotient is 4
Modulus is 3

This is on one line
This on another
This shows      how      tabs      work

C:\Student Data\Unit_02>_

```

Exhibit 2-14: Output of the DemoVariables7 program

Do it!

## B-9: Discussing escape sequences

### Questions and answers

- 1 An escape sequence always begins with a \_\_\_\_\_.
  - A 'e'
  - B forward slash
  - C backslash**
  - D equals sign
- 2 You type constant character values in \_\_\_\_\_ quotes.
 

**single**
- 3 You type string constants between \_\_\_\_\_ quotation marks.
 

**double**

## ASCII and Unicode

Explanation



The characters used in the Java programming language are represented in *Unicode*, which is a 16-bit coding scheme for characters. For example, the letter A actually is stored in computer memory as a set of 16 zeros and ones as 0000 0000 0100 0001. The space inserted after each set of four digits is for readability.

Programmers often use a shorthand notation called *hexadecimal*, or base 16, because 16-digit numbers are difficult to read. In hexadecimal shorthand, 0000 becomes 0; 0100 becomes 4, and 0001 becomes 1, the letter A is represented in hexadecimal as 0041. You tell the compiler to treat the four-digit hexadecimal 0041 as a single character by preceding it with the `\u` escape sequence, therefore, there are two ways to store the character A:

```
char letter = 'A';
char letter = '\u0041';
```

**Note:** For more information about Unicode, go to <http://www.unicode.org>.

The second option that uses hexadecimal is more difficult and confusing than the first method. It is not recommended that you store letters of the alphabet by using the hexadecimal method, however, there are some interesting values you can produce using the Unicode format. For example, the sequence `'\u0007'` is a bell that produces a noise if you send it to output. Letters from foreign alphabets that use characters instead of letters (Greek, Hebrew, Chinese, and so on) and other special symbols (foreign currency symbols, mathematical symbols, geometric shapes, and so on) are available using Unicode, but not on a standard keyboard. It is important that you know how to use Unicode characters.

The most widely used character set is *ASCII*. (American Standard Code for Information Interchange.) There are 128 characters in the ASCII character set, with their decimal code or numerical code and equivalent character representation. The first 32 characters and the last character are control characters and are nonprintable. You can enter these characters by holding down [Ctrl] and pressing a letter on the keyboard. For example, The Tab key or `^I`(Ctrl I) produces a character 9, that produces a hard tab when pressed.

The relationship of ASCII and Unicode is that by adding eight zeros to any ASCII character gives the character's value in Unicode. The ASCII values are important because you can use them to show nonprintable characters, such as a carriage return, in decimal codes. ASCII codes are often used when sorting numbers and strings, so when you need to sort characters in ascending order, numbers beginning with decimal 48 or 0 are sorted first, then capital letters starting with decimal 65 or A, and then lowercase letters starting with decimal code 97 or a.

*Do it!***B-10: Discussing ASCII and Unicode****Questions and answers**

- 1 The 16-bit coding scheme employed by the Java programming language is called \_\_\_\_\_.  
**A** Unicode  
B ASCII  
C EBCDIC  
D hexadecimal
- 2 How many characters are there in the ASCII character set?  
A 256  
**B** 128  
C 127  
D 512
- 3 Hexadecimal is base \_\_\_\_\_.  
A 8  
**B** 16  
C 2  
D 10

## Unit summary: Using data in a program

### Topic A

In this topic, you learned that data can be categorized as a **variable** or **constant**. You learned that data is **constant** when it can't be changed after a program is compiled; and the data is **variable** when it might change. You also learned about the **primitive data types** and how to **declare variables** in a Java program.

### Topic B

In this topic, you learned the eight primitive data types, **boolean**, **byte**, **char**, **double**, **float**, **int**, **long**, and **short**, in detail. You learned that **operator precedence** is the order in which parts of a mathematical expression are evaluated. You also learned about the two character sets: **ASCII** and **Unicode**.

### Review questions

- 1 The assignment operator in the Java programming language is \_\_\_\_\_.  
A =  
B ==  
C :=  
D ::
- 2 Which of the following values can you assign to a variable of type `int`?  
A 0  
B 98.6  
C 'S'  
D 5,000,000,000,000
- 3 Boolean variables can hold \_\_\_\_\_.  
A any character  
B any whole number  
C any decimal number  
D the value true or false
- 4 The "equal to" comparison operator is \_\_\_\_\_.  
A =  
B ==  
C !=  
D !!

- 5 If you attempt to add a `float`, an `int`, and a `byte`, the result will be \_\_\_\_\_.
- A float**
  - B `int`
  - C `byte`
  - D error message
- 6 Which assignment is correct?
- A `char aChar = 5;`
  - B `char aChar = "W";`
  - C `char aChar = '+';`**
  - D Two of the preceding answers are correct.

### Independent practice activity

Sample solutions for the following activities are provided in the current unit's solutions folder.

- 1 Write a program that declares variables to represent the length and width of a room in feet, and the price of carpeting per square foot in dollars and cents. Use `Carpet` as the class name. Assign appropriate values to the variables. Compute and display, with explanatory text, the cost of carpeting the room. Save the program as **`Carpet.java`** in the current unit folder.
- 2 Write a program that declares variables to represent the length and width of a room in feet, and the price of carpeting per square yard in dollars and cents. Use `Yards` as the class name. Assign the value 25 to the length variable and the value 42 to the width variable. Compute and display the cost of carpeting the room. (Hint: There are nine square feet in one square yard.) Save the program as **`Yards.java`**.
- 3 Write a program that declares a `minutes` variable that represents minutes worked on a job, and assign a value. Use `Time` as the class name. Display the value in hours and minutes. For example, 197 minutes becomes 3 hours and 17 minutes. Save the program as **`Time.java`**.
- 4 Write a program that declares variables to hold your three initials. Display the three initials with a period following each one, as in J.M.F. Save the program as **`Initials.java`**.
- 5 Write a program that displays `FirstName`, `LastName`, `Address`, and `Phone` on one line of output, and your first name, last name, address, and phone number on the second line. Make sure that your data lines up with the headings. Save the program as **`Escape.java`**.

**PREVIEW**

**NOT FOR PRINTING OR INSTRUCTIONAL USE**