Programming Languages: Java

Lecture 1
Introduction to Java

Instructor: Omer Boyaci

© 1992-2007 Pearson Education, Inc. All rights reserved.

Outline

- Course Information
- History of Java
- Introduction
- First Program in Java: Printing a Line of Text
- Modifying Our First Java Program
- Displaying Text with `printf`
- Another Java Application: Adding Integers
- Memory Concepts
- Arithmetic
- Decision Making: Equality and Relational Operators
- Introduction to Object-oriented Programming

© 1992-2007 Pearson Education, Inc. All rights reserved.

Course Information

- Six Lectures
- Teaches “Java Standart Edition 6”
- No midterm or final
- Six assignments (5,10,15,20,25,25)
- http://www.omerboyaci.com/
- Textbook
  - Java How to Program, 8th Edition, Deitel & Deitel

© 1992-2007 Pearson Education, Inc. All rights reserved.

Introduction

- **Java Standard Edition (Java SE) 6**
- **Sun's implementation called the Java Development Kit (JDK)**
- **Object-Oriented Programming**
- **Java is language of choice for networked applications**
- **Open Source**
- **Write Once Run Everywhere**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Machine Languages, Assembly Languages and High-Level Languages

- Machine language
  - "Natural language" of computer component
  - Machine dependent
- Assembly language
  - English-like abbreviations represent computer operations
  - Translator programs (assemblers) convert to machine language
- High-level language
  - Allows for writing more "English-like" instructions
    - Contains commonly used mathematical operations
  - Compiler converts to machine language
- Interpreter
  - Execute high-level language programs without compilation

© 1992-2007 Pearson Education, Inc. All rights reserved.

History of Java

- **Java**
  - **Originally for intelligent consumer-electronic devices**
  - **Then used for creating web pages with dynamic content**
  - **Now also used to:**
    - **Develop large-scale enterprise applications**
    - **Enhance web server functionality**
    - **Provide applications for consumer devices (cell phones, etc.)**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Java Platform

Servers & enterprise computers

Servers & personal computers

High-end PDAs TV set-top boxes Embedded devices

Mobile phones & entry-level PDAs

Smart cards

Optional Packages

Java 2 Platform, Enterprise Edition (J2EE)

JVM™

Optional Packages

Java 2 Platform, Standard Edition (J2SE)

JVM

Optional Packages

Personal Profile

Personal Basis Profile

Foundation Profile

CDC

JVM

Optional Packages

MIDP

CLDC

KVM

Java Card

Card VM

Java Platform, Micro Edition (Java ME)

© 1992-2007 Pearson Education, Inc. All rights reserved.

Java Standart Edition (SE)

Java™ SE Platform at a Glance

JDK (covers all layers) · JRE (Deployment Technologies through Java Virtual Machine) · Java SE API (User Interface Toolkits through lang and util Base Libraries)

Layer	Components
Java Language	Java Language
Tools & Tool APIs	java, javac, javadoc, apt, jar, javap, JPDA, JConsole, Java VisualVM
	Security, Int'l, RMI, IDL, Deploy, Monitoring, Troubleshoot, Scripting, JVM TI
Deployment Technologies	Deployment, Java Web Start, Java Plug-in
User Interface Toolkits	AWT, Swing, Java 2D
	Accessibility, Drag n Drop, Input Methods, Image I/O, Print Service, Sound
Integration Libraries	IDL, JDBC, JNDI, RMI, RMI-IIOP
Other Base Libraries	Beans, Intl Support, Input/Output, JMX, JNI, Math
	Networking, Override Mechanism, Security, Serialization, Extension Mechanism, XML JAXP
lang and util Base Libraries	lang and util, Collections, Concurrency Utilities, JAR, Logging, Management
	Preferences API, Ref Objects, Reflection, Regular Expressions, Versioning, Zip, Instrumentation
Java Virtual Machine	Java Hotspot Client VM, Java Hotspot Server VM
Platforms	Solaris, Linux, Windows, Other

© 1992-2007 Pearson Education, Inc. All rights reserved.

Java Enterprise Edition (EE)

geared toward large-scale distributed applications and web applications

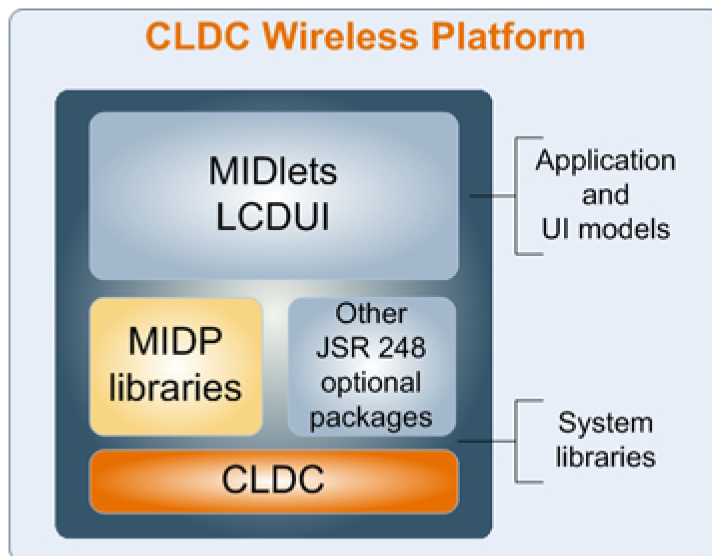
- **Enterprise JavaBeans (EJB)**
- **Servlets**
- **Java Server Pages (JSP)**
- **Java Server Faces (JSF)**
- **JavaMail**
- **Java Transaction API (JTA)**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Java Micro Edition (ME)

geared toward applications for small, memory constrained devices

- **Midlets**
  - **Google Maps Mobile**
  - **Opera Mini**



© 1992-2007 Pearson Education, Inc. All rights reserved.

Java Class Libraries

- **Java programs consist of classes**
  - **Include methods that perform tasks**
    - **Return information after task completion**
- **Java provides class libraries**
  - **Known as Java APIs (Application Programming Interfaces)**
- **To use Java effectively, you must know**
  - **Java programming language**
  - **Extensive class libraries**

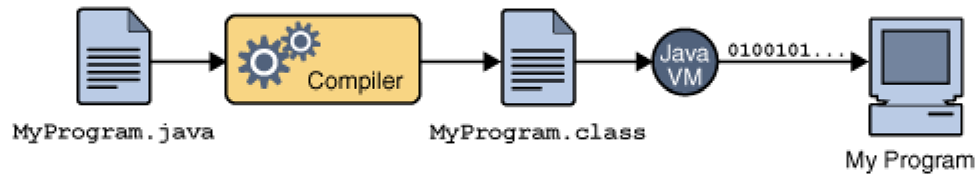
© 1992-2007 Pearson Education, Inc. All rights reserved.

Use Java API classes

- Improve program performance
- Shorten program development time
- Prevent software bugs
- Improve program portability

© 1992-2007 Pearson Education, Inc. All rights reserved.

Typical Java Development Environment



- Java programs go through five phases
  - Edit
    - Programmer writes program using an editor; stores program on disk with the `.java` file name extension
  - Compile
    - Use javac (the Java compiler) to create bytecodes from source code program; bytecodes stored in `.class` files
  - Load
    - Class loader reads bytecodes from `.class` files into memory
  - Verify
    - Bytecode verifier examines bytecodes to ensure that they are valid and do not violate security restrictions
  - Execute
    - Java Virtual Machine (JVM) uses a combination of interpretation and just-in-time compilation to translate bytecodes into machine language

© 1992-2007 Pearson Education, Inc. All rights reserved.

Phase 1: Edit — Editor ↔ Disk

Program is created in an editor and stored on disk in a file ending with `.java`.

Phase 2: Compile — Compiler ↔ Disk

Compiler creates bytecodes and stores them on disk in a file ending with `.class`.

Phase 3: Load — Disk → Class Loader → Primary Memory

Class loader reads `.class` files containing bytecodes from disk and puts those bytecodes in memory.

Phase 4: Verify — Bytecode Verifier ↔ Primary Memory

Bytecode verifier confirms that all bytecodes are valid and do not violate Java's security restrictions.

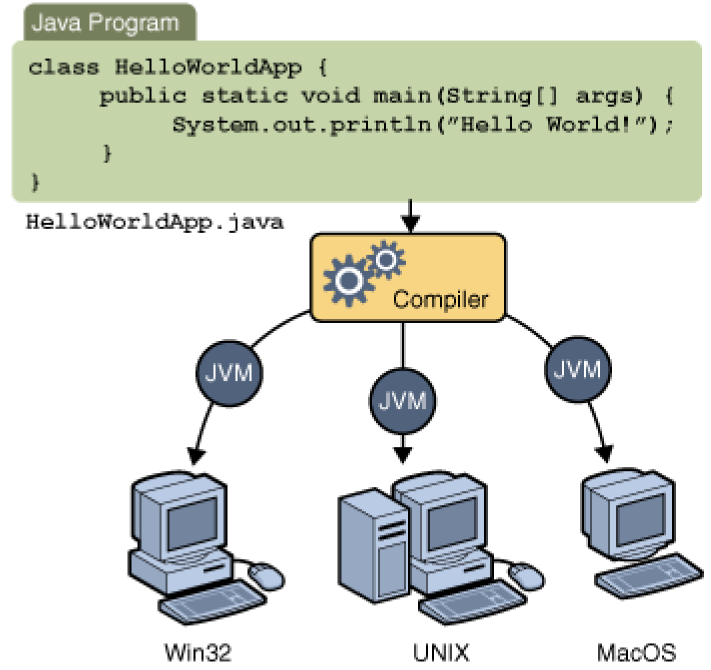
Phase 5: Execute — Java Virtual Machine (JVM) ↔ Primary Memory

To execute the program, the JVM reads bytecodes and translates them into a language that the computer can understand. As the program executes, it may store data values in primary memory.

Fig. 1.1 | Typical Java development environment.

© 1992-2007 Pearson Education, Inc. All rights reserved.

Through the Java VM, the same application is capable of running on multiple platforms.



© 1992-2007 Pearson Education, Inc. All rights reserved.

First Program in Java: Printing a Line of Text

- **Application**
  - **Executes when you use the `java` command to launch the Java Virtual Machine (JVM)**
- **Sample program**
  - **Displays a line of text**
  - **Illustrates several important Java language features**

© 1992-2007 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 2.1: Welcome1.java
2  // Text-printing program.
3
4  public class Welcome1
5  {
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9        System.out.println( "Welcome to Java Programming!" );
10
11    } // end method main
12
13 } // end clazss Welcome1
```

```
Welcome to Java Programming!
```

Welcome1.java

© 1992-2007 Pearson Education, Inc. All rights reserved.

First Program in Java: Printing a Line of Text (Cont.)

```
1   // Fig. 2.1: Welcome1.java
```

- Comments start with: `//`
  - Comments ignored during program execution
  - Document and describe code
  - Provides code readability
- Traditional comments: `/* ... */`

```
/* This is a traditional
   comment. It can be
   split over many lines */
```

```
2   // Text-printing program.
```

- Another line of comments
- Note: line numbers not part of program, added for reference

© 1992-2007 Pearson Education, Inc. All rights reserved.

First Program in Java: Printing a Line of Text (Cont.)

```
3
```

- Blank line
  - Makes program more readable
  - Blank lines, spaces, and tabs are white-space characters
    - Ignored by compiler

```
4   public class Welcome1
```

- Begins class declaration for class `Welcome1`
  - Every Java program has at least one user-defined class
  - Keyword: words reserved for use by Java
    - `class` keyword followed by class name
  - Naming classes: capitalize every word
    - `SampleClassName`

© 1992-2007 Pearson Education, Inc. All rights reserved.

First Program in Java: Printing a Line of Text (Cont.)

```
4   public class Welcome1
```

– Java identifier
  - Series of characters consisting of letters, digits, underscores (_) and dollar signs ($)
  - Does not begin with a digit, has no spaces
  - Examples: `Welcome1`, `$value`, `_value`, `button7`
    – `7button` is invalid
  - Java is case sensitive (capitalization matters)
    – `a1` and `A1` are different

© 1992-2007 Pearson Education, Inc. All rights reserved.

First Program in Java: Printing a Line of Text (Cont.)

```
4   public class Welcome1
```

- Saving files
  - File name must be class name with `.java` extension
  - `Welcome1.java`

```
5   {
```

- Left brace **{**
  - Begins body of every class
  - Right brace ends declarations (line 13)

© 1992-2007 Pearson Education, Inc. All rights reserved.

First Program in Java: Printing a Line of Text (Cont.)

```
7       public static void main( String args[] )
```

- Part of every Java application
  - Applications begin executing at `main`
    - Parentheses indicate `main` is a method
    - Java applications contain one or more methods
  - Exactly one method must be called `main`
- Methods can perform tasks and return information
  - `void` means `main` returns no information
  - For now, mimic `main`'s first line

```
8       {
```

- Left brace begins body of method declaration
  - Ended by right brace } (line 11)

© 1992-2007 Pearson Education, Inc. All rights reserved.

First Program in Java: Printing a Line of Text (Cont.)

```
9        System.out.println( "Welcome to Java Programming!" );
```

- Instructs computer to perform an action
  - Prints string of characters
    - String – series of characters inside double quotes
  - White-spaces in strings are not ignored by compiler
- `System.out`
  - Standard output object
  - Print to command window (i.e., MS-DOS prompt)
- Method `System.out.println`
  - Displays line of text
- This line known as a statement
  - Statements must end with semicolon `;`

© 1992-2007 Pearson Education, Inc. All rights reserved.

First Program in Java: Printing a Line of Text (Cont.)

```
11     } // end method main
```

– **Ends method declaration**

```
13  } // end class Welcome1
```

– **Ends class declaration**
– **Can add comments to keep track of ending braces**

© 1992-2007 Pearson Education, Inc. All rights reserved.

First Program in Java: Printing a Line of Text (Cont.)

- Compiling a program
  - Open a command prompt window, go to directory where program is stored
  - Type `javac Welcome1.java`
  - If no syntax errors, `Welcome1.class` created
    - Has bytecodes that represent application
    - Bytecodes passed to JVM
- system's *PATH* environment variable for java and javac

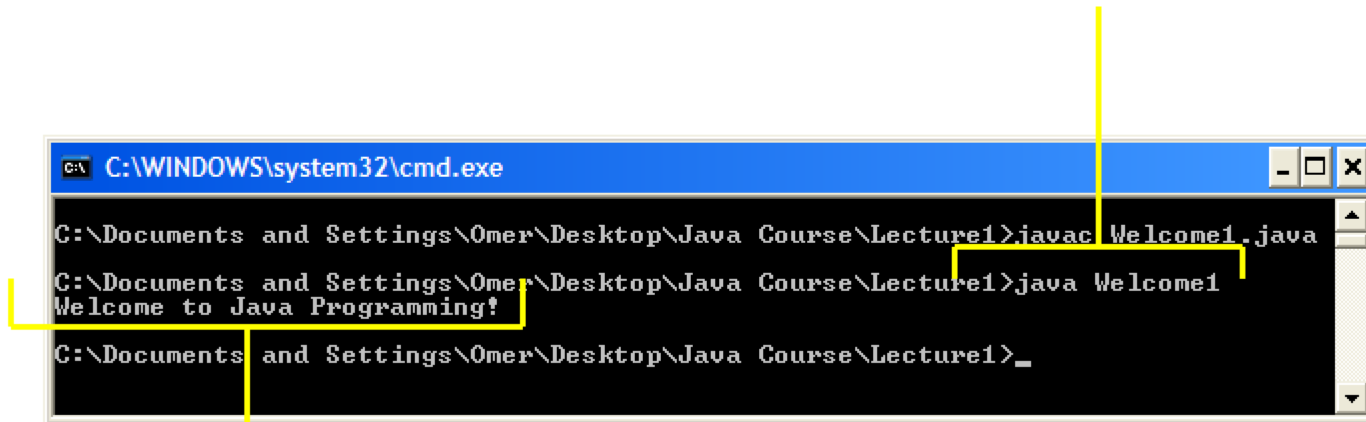
© 1992-2007 Pearson Education, Inc. All rights reserved.

First Program in Java: Printing a Line of Text (Cont.)

- Executing a program
  - Type `java Welcome1`
    - Launches JVM
    - JVM loads `.class` file for class `Welcome1`
    - `.class` extension omitted from command
    - JVM calls method `main`

© 1992-2007 Pearson Education, Inc. All rights reserved.

You type this command to execute the application



The program outputs

Welcome to Java Programming!

Executing `Welcome1` in a Microsoft Windows XP Command Prompt **window.**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Modifying Our First Java Program

- **Modify example in Fig. 2.1 to print same contents using different code**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Modifying Our First Java Program (Cont.)

- Modifying programs
  - `Welcome2.java` (Fig. 2.3) produces same output as `Welcome1.java` (Fig. 2.1)
  - Using different code

```
9         System.out.print( "Welcome to " );
10        System.out.println( "Java Programming!" );
```

  - Line 9 displays “Welcome to ” with cursor remaining on printed line
  - Line 10 displays “Java Programming! ” on same line with cursor on next line

© 1992-2007 Pearson Education, Inc. All rights reserved.

```java
1  // Fig. 2.3: Welcome2.java
2  // Printing a line of text with multiple statements.
3
4  public class Welcome2
5  {
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9        System.out.print( "Welcome to " );
10       System.out.println( "Java Programming!" );
11
12    } // end method main
13
14 } // end class Welcome2
```

System.out.print keeps the cursor on the same line, so System.out.println continues on the same line.

```
Welcome to Java Programming!
```

Outline

1. Comments

2. Blank line

3. Begin class Welcome2

3.1 Method main

4. Method System.out.print

4.1 Method System.out.println

5. end main, Welcome2

Program Output

© 1992-2007 Pearson Education, Inc. All rights reserved.

Modifying Our First Java Program (Cont.)

- Escape characters
  - Backslash (\ )
  - Indicates special characters to be output
- Newline characters (`\n`)
  - Interpreted as "special characters" by methods `System.out.print` and `System.out.println`
  - Indicates cursor should be at the beginning of the next line
  - `Welcome3.java` (Fig. 2.4)

```
9         System.out.println( "Welcome\nto\nJava\nProgramming!" );
```

  - Line breaks at `\n`

© 1992-2007 Pearson Education, Inc. All rights reserved.

```
1  // Fig. 2.4: Welcome3.java
2  // Printing multiple lines of text with a single statement.
3
4  public class Welcome3
5  {
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9        System.out.println( "Welcome\nto\nJava\nProgramming!" );
10
11    } // end method main
12
13 } // end class Welcome3
```

```
Welcome
to
Java
Programming!
```

A new line begins after each `\n` escape sequence is output.

Outline

`Welcome3.java`

1. `main`

2. `System.out.println` (uses `\n` for new line)

Program Output

© 1992-2007 Pearson Education, Inc. All rights reserved.

Escape sequence	Description
`\n`	Newline. Position the screen cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor at the beginning of the current line—do not advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double-quote character. For example, `System.out.println( "\"in quotes\"" );` displays `"in quotes"`

Fig. 2.5 | **Some common escape sequences.**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Displaying Text with `printf`

- `System.out.printf`
  - Feature added in Java SE 5.0
  - Displays formatted data

```
9          System.out.printf( "%s\n%s\n",
10            "Welcome to", "Java Programming!" );
```

  - Format string
    - Fixed text
    - Format specifier – placeholder for a value
  - Format specifier `%s` – placeholder for a string

© 1992-2007 Pearson Education, Inc. All rights reserved.

Outline

```
1  // Fig. 2.6: Welcome4.java
2  // Printing multiple lines in a dialog box.
3
4  public class Welcome4
5  {
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9        System.out.printf( "%s\n%s\n",
10          "Welcome to", "Java Programming!" );
11
12    } // end method main
13
14 } // end class Welcome4
```

System.out.printf displays formatted data.

```
Welcome to
Java Programming!
```

- Welcome4.java
- main
- printf
- Program output

© 1992-2007 Pearson Education, Inc. All rights reserved.

Another Java Application: Adding Integers

- Upcoming program
  - Use `Scanner` to read two integers from user
  - Use `printf` to display sum of the two values
  - Use packages

© 1992-2007 Pearson Education, Inc. All rights reserved.

Outline

Addition.java

(1 of 2)

```
1  // Fig. 2.7: Addition.java
2  // Addition program that displays the sum of two numbers.
3  import java.util.Scanner; // program uses class Scanner
4
5  public class Addition
6  {
7     // main method begins execution of Java application
8     public static void main( String args[] )
9     {
10       // create Scanner to obtain input from command window
11       Scanner input = new Scanner( System.in );
12
13       int number1; // first number to add
14       int number2; // second number to add
15       int sum; // sum of number1 and number2
16
17       System.out.print( "Enter first integer: " ); // prompt
18       number1 = input.nextInt(); // read first number from user
19
```

`import` declaration imports class `Scanner` from package `java.util`.

Declare and initialize variable `input`, which is a `Scanner`.

Declare variables `number1`, `number2` and `sum`.

Read an integer from the user and assign it to `number1`.

© 1992-2007 Pearson Education, Inc. All rights reserved.

Outline

```
20        System.out.print( "Enter second integer: " ); // prompt
21        number2 = input.nextInt(); // read second number from user
22
23        sum = number1 + number2; // add numbers
24
25        System.out.printf( "Sum is %d\n", sum ); // d
26
27     } // end method main
28
29  } // end class Addition
```

Read an integer from the user and assign it to `number2`.

Calculate the sum of the variables `number1` and `number2`, assign result to `sum`.

Display the sum using formatted output.

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Two integers entered by the user.

- dition.java
- of 2)
- 4. Addition
- 5. `printf`

```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\Omer\Desktop\Java Course\Lecture1>javac Addition.java

C:\Documents and Settings\Omer\Desktop\Java Course\Lecture1>java Addition
Enter first integer: 23
Enter second integer: 17
Sum is 40

C:\Documents and Settings\Omer\Desktop\Java Course\Lecture1>
```

© 1992-2007 Pearson Education, Inc. All rights reserved.

Another Java Application: Adding Integers (Cont.)

```
3   import java.util.Scanner;  // program uses class Scanner
```

– `import` declarations
  - Used by compiler to identify and locate classes used in Java programs
  - Tells compiler to load class `Scanner` from `java.util` package

```
5   public class Addition
6      {
```

– Begins `public` class `Addition`
  - Recall that file name must be `Addition.java`

– Lines 8-9: begin `main`

© 1992-2007 Pearson Education, Inc. All rights reserved.

Another Java Application: Adding Integers (Cont.)

```
10      // create Scanner to obtain input from command window
11      Scanner input = new Scanner( System.in );
```

- Variable Declaration Statement
- Variables
  - Location in memory that stores a value
    - Declare with name and type before use
  - `Input` is of type `Scanner`
    - Enables a program to read data for use
  - Variable name: any valid identifier
- Declarations end with semicolons `;`
- Initialize variable in its declaration
  - Equal sign
  - Standard input object
    - `System.in`

© 1992-2007 Pearson Education, Inc. All rights reserved.

Another Java Application: Adding Integers (Cont.)

```
13        int number1; // first number to add
14        int number2; // second number to add
15        int sum; // sum of number 1 and number 2
```

– Declare variable `number1`, `number2` and `sum` of type `int`
  - `int` holds integer values (whole numbers): i.e., `0`, `-4`, `97`
  - Types `float` and `double` can hold decimal numbers
  - Type `char` can hold a single character: i.e., x, $, \n, 7
  - `int`, `float`, `double` and `char` are primitive types

– Can add comments to describe purpose of variables

```
      int number1, // first number to add
          number2, // second number to add
          sum; // sum of number1 and number2
```

– Can declare multiple variables of the same type in one declaration

– Use comma-separated list

© 1992-2007 Pearson Education, Inc. All rights reserved.

Another Java Application: Adding Integers (Cont.)

```
17        System.out.print( "Enter first integer: " ); // prompt
```

- Message called a prompt - directs user to perform an action
- Package `java.lang`

```
18        number1 = input.nextInt(); // read first number from user
```

- Result of call to `nextInt` given to `number1` using assignment operator `=`
  - Assignment statement
  - `=` binary operator - takes two operands
    - Expression on right evaluated and assigned to variable on left
  - Read as: `number1` gets the value of `input.nextInt()`

© 1992-2007 Pearson Education, Inc. All rights reserved.

Another Java Application: Adding Integers (Cont.)

```
20        System.out.print( "Enter second integer: " ); // prompt
```

- Similar to previous statement
  - Prompts the user to input the second integer

```
21        number2 = input.nextInt(); // read second number from user
```

- Similar to previous statement
  - Assign variable number2 to second integer input

```
23        sum = number1 + number2; // add numbers
```

- Assignment statement
  - Calculates sum of `number1` and `number2` (right hand side)
  - Uses assignment operator = to assign result to variable `sum`
  - Read as: `sum` gets the value of `number1 + number2`
  - `number1` and `number2` are operands

© 1992-2007 Pearson Education, Inc. All rights reserved.

Another Java Application: Adding Integers (Cont.)

```
25          System.out.printf( "Sum is %d\n " , sum ); // display sum
```

- Use `System.out.printf` to display results
- Format specifier `%d`
  - Placeholder for an `int` value

```
            System.out.printf( "Sum is %d\n " , ( number1 + number2 ) );
```

- Calculations can also be performed inside `printf`
- Parentheses around the expression `number1 + number2` are not required

© 1992-2007 Pearson Education, Inc. All rights reserved.

Memory Concepts

- Variables
  - Every variable has a name, a type, a size and a value
    - Name corresponds to location in memory
  - When new value is placed into a variable, replaces (and destroys) previous value
  - Reading variables from memory does not change them

© 1992-2007 Pearson Education, Inc. All rights reserved.

Fig. 2.8 | Memory location showing the name and value of variable `number1`.

© 1992-2007 Pearson Education, Inc. All rights reserved.

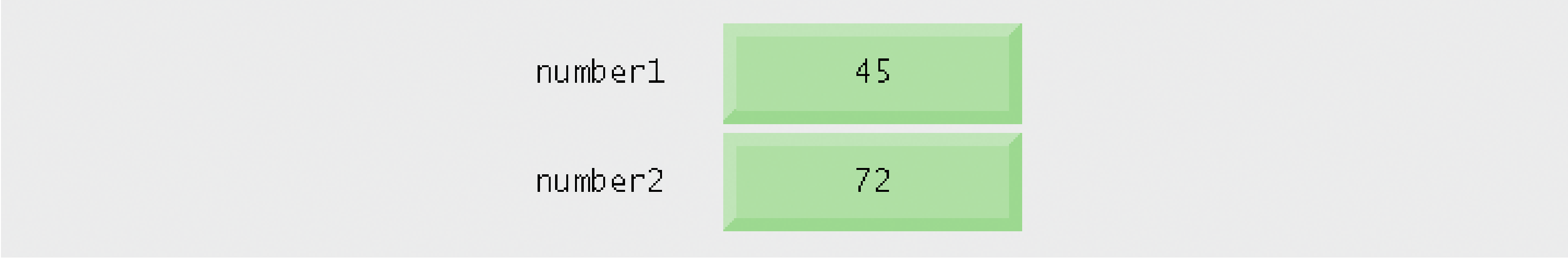


Fig. 2.9 | Memory locations after storing values for `number1` and `number2`.

© 1992-2007 Pearson Education, Inc. All rights reserved.

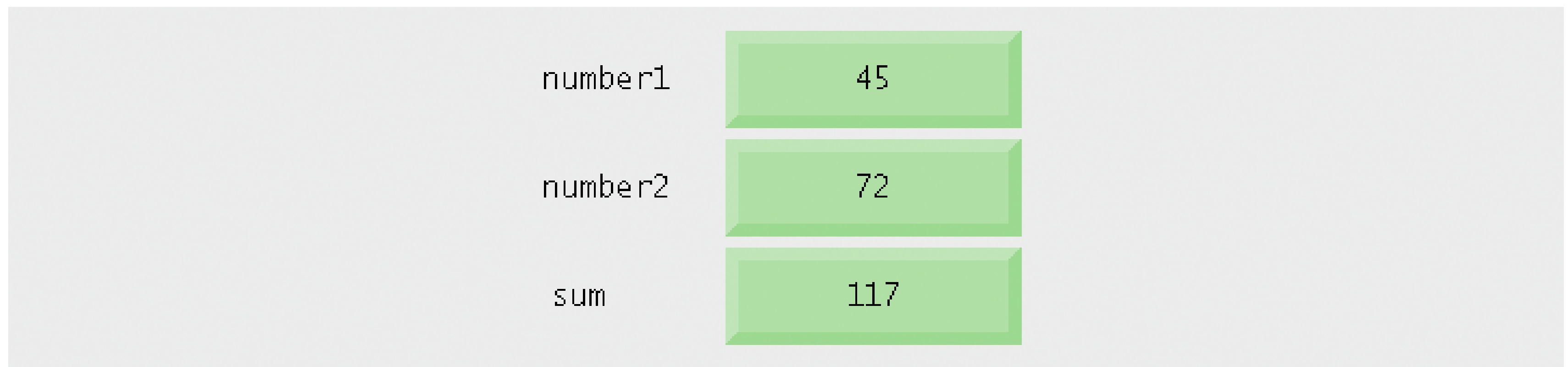


Fig. 2.10 | Memory locations after calculating and storing the sum of `number1` and `number2`.

© 1992-2007 Pearson Education, Inc. All rights reserved.

Arithmetic

- Arithmetic calculations used in most programs
  - Usage
    - * for multiplication
    - / for division
    - % for remainder
    - +, -
  - Integer division truncates remainder

    7 / 5 evaluates to 1
  - Remainder operator % returns the remainder

    7 % 5 evaluates to 2

© 1992-2007 Pearson Education, Inc. All rights reserved.

Java operation	Arithmetic operator	Algebraic expression	Java expression
Addition	+	$f+7$	`f + 7`
Subtraction	–	$p-c$	`p - c`
Multiplication	*	bm	`b * m`
Division	/	x/y or $\frac{x}{y}$ or $x \div y$	`x / y`

Fig. 2.11 | Arithmetic operators.

© 1992-2007 Pearson Education, Inc. All rights reserved.

Arithmetic (Cont.)

- **Operator precedence**
  - **Some arithmetic operators act before others (i.e., multiplication before addition)**
    - **Use parenthesis when needed**
  - **Example: Find the average of three variables `a`, `b` and `c`**
    - **Do not use:** `a + b + c / 3`
    - **Use:** `( a + b + c ) / 3`

© 1992-2007 Pearson Education, Inc. All rights reserved.

Operator(s)	Operation(s)	Order of evaluation (precedence)
* / %	Multiplication Division Remainder	Evaluated first. If there are several operators of this type, they are evaluated from left to right.
+ -	Addition Subtraction	Evaluated next. If there are several operators of this type, they are evaluated from left to right.

Fig. 2.12 | **Precedence of arithmetic operators.**

© 1992-2007 Pearson Education, Inc. All rights reserved.

```
Step 1.   y = 2 * 5 * 5 + 3 * 5 + 7;        (Leftmost multiplication)
              2 * 5 is 10

Step 2.   y = 10 * 5 + 3 * 5 + 7;           (Leftmost multiplication)
              10 * 5 is 50

Step 3.   y = 50 + 3 * 5 + 7;               (Multiplication before addition)
                   3 * 5 is 15

Step 4.   y = 50 + 15 + 7;                  (Leftmost addition)
              50 + 15 is 65

Step 5.   y = 65 + 7;                       (Last addition)
              65 + 7 is 72

Step 6.   y = 72                            (Last operation—place 72 in y)
```

Fig. 2.13 | Order in which a second-degree polynomial is evaluated.

© 1992-2007 Pearson Education, Inc. All rights reserved.

Decision Making: Equality and Relational Operators

- Condition
  - Expression can be either `true` or `false`
- `if` statement
  - Simple version in this section, more detail later
  - If a condition is `true`, then the body of the `if` statement executed
  - Control always resumes after the `if` statement
  - Conditions in `if` statements can be formed using equality or relational operators (next slide)

© 1992-2007 Pearson Education, Inc. All rights reserved.

Standard algebraic equality or relational operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Fig. 2.14 | **Equality and relational operators.**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Outline

```
1  // Fig. 2.15: Comparison.java
2  // Compare integers using if statements, relational operators
3  // and equality operators.
4  import java.util.Scanner; // program uses class Scanner
5
6  public class Comparison
7  {
8     // main method begins execution of Java application
9     public static void main( String args[] )
10    {
11       // create Scanner to obtain input from command window
12       Scanner input = new Scanner( System.in );
13
14       int number1; // first number to compare
15       int number2; // second number to compare
16
17       System.out.print( "Enter first integer: " ); // prompt
18       number1 = input.nextInt(); // read first number from user
19
20       System.out.print( "Enter second integer: " ); // prompt
21       number2 = input.nextInt(); // read second number from user
22
23       if ( number1 == number2 )
24          System.out.printf( "%d == %d\n", number1, number2 );
25
26       if ( number1 != number2 )
27          System.out.printf( "%d != %d\n", number1, number2 );
28
29       if ( number1 < number2 )
30          System.out.printf( "%d < %d\n", number1,
```

Test for equality, display result using `printf`.

Compares two numbers using relational operator <.

Comparison.java

(1 of 2)

1. Class Comparison

1.1 main

1.2 Declarations

1.3 Input data (nextInt)

1.4 Compare two inputs using if statements

© 1992-2007 Pearson Education, Inc. All rights reserved.

Outline

```
31
32        if ( number1 > number2 )
33           System.out.printf( "%d > %d\n", number1, number2 );
34
35        if ( number1 <= number2 )
36           System.out.printf( "%d <= %d\n", number1, number2 );
37
38        if ( number1 >= number2 )
39           System.out.printf( "%d >= %d\n", number1, number2 );
40
41     } // end method main
42
43  } // end class Comparison
```

Compares two numbers using relational operators >, <= and >=.

Comparison.java

(2 of 2)

Program output

```
Enter first integer: 777
Enter second integer: 777
777 == 777
777 <= 777
777 >= 777
```

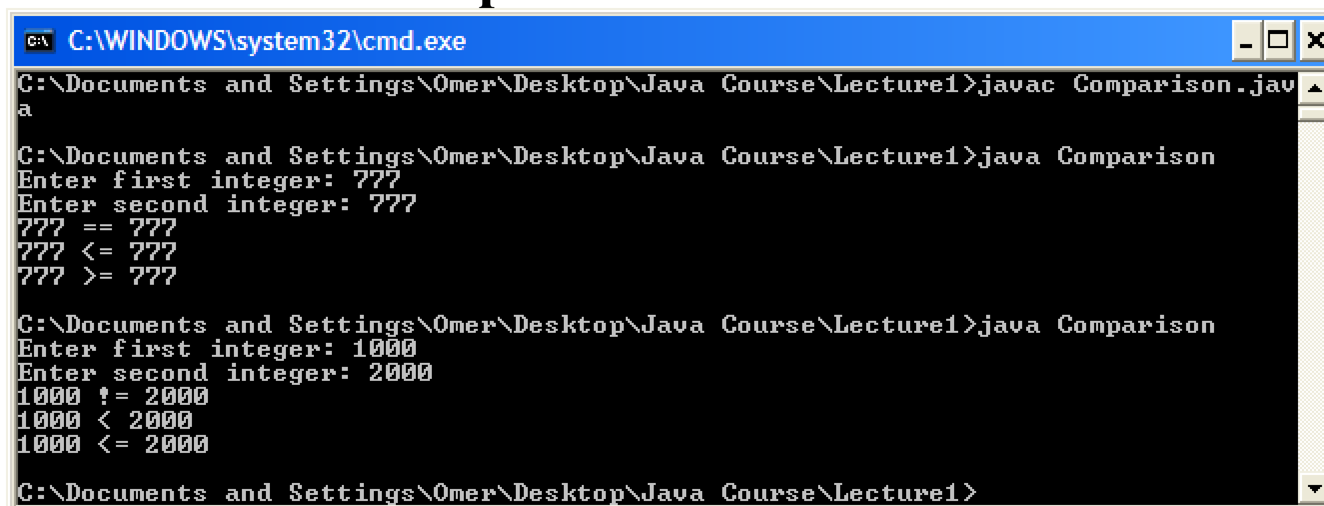
```
Enter first integer: 1000
Enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000
```

```
Enter first integer: 2000
Enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000
```

© 1992-2007 Pearson Education, Inc. All rights reserved.

Decision Making: Equality and Relational Operators (Cont.)

- Line 6: begins class `Comparison` declaration
- Line 12: declares Scanner variable input and assigns it a Scanner that inputs data from the standard input
- Lines 14-15: declare `int` variables
- Lines 17-18: prompt the user to enter the first integer and input the value
- Lines 20-21: prompt the user to enter the second integer and input the value



© 1992-2007 Pearson Education, Inc. All rights reserved.

Decision Making: Equality and Relational Operators (Cont.)

```
23         if ( number1 == number2 )
24            System.out.printf( "%d == %d\n", number1, number2 );
```

- `if` statement to test for equality using (==)
  - If variables equal (condition true)
    - Line 24 executes
  - If variables not equal, statement skipped
  - No semicolon at the end of line 23
  - Empty statement
    - No task is performed
- Lines 26-27, 29-30, 32-33, 35-36 and 38-39
  - Compare `number1` and `number2` with the operators `!=`, `<`, `>`, `<=` and `>=`, respectively

© 1992-2007 Pearson Education, Inc. All rights reserved.

Operators	Associativity	Type
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
=	right to left	assignment

Fig. 2.16 | **Precedence and associativity of operations discussed.**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Object-oriented Programming

- **Objects**
  - **Reusable software components that model real-world items**
  - **Look all around you**
    - **People, animals, plants, cars, etc.**
  - **Attributes**
    - **Size, shape, color, weight, etc.**
  - **Behaviors**
    - **Babies cry, crawl, sleep, etc.**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Object-oriented Programming

- Object-oriented design (OOD)
  - Models software in terms similar to those used to describe real-world objects
  - Class relationships
  - Inheritance relationships
  - Models communication among objects
  - Encapsulates attributes and operations (behaviors)
    - Information hiding
    - Communication through well-defined interfaces
- Object-oriented language
  - Programming in object-oriented languages is called *object-oriented programming* (*OOP*)
  - Java

© 1992-2007 Pearson Education, Inc. All rights reserved.

Object-oriented Programming

- Classes are to objects as blueprints are to houses
- Associations
  - Relationships between classes
- Packaging software in classes facilitates reuse

© 1992-2007 Pearson Education, Inc. All rights reserved.

Object-oriented Programming

- Object-Oriented Analysis and Design (OOA/D)
  - Essential for large programs
  - Analyze program requirements, then develop a design
  - UML
    - Unified Modeling Language
    - Standard for designing object-oriented systems

© 1992-2007 Pearson Education, Inc. All rights reserved.

Object-oriented Programming

- **History of the UML**
  - **Need developed for process with which to approach OOA/D**
  - **Brainchild of Booch, Rumbaugh and Jacobson**
  - **Object Management Group (OMG) supervised**
  - **Version 2 is current version**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Object-oriented Programming

- UML
  - Graphical representation scheme
  - Enables developers to model object-oriented systems
  - Flexible and extensible

© 1992-2007 Pearson Education, Inc. All rights reserved.

Control Statements

© 1992-2007 Pearson Education, Inc. All rights reserved.

Outline

- Introduction
- Algorithms
- Pseudocode
- Control Structures
- `if` Single-Selection Statement
- `if…else` Double-Selection Statement
- `while` Repetition Statement
- Formulating Algorithms: Counter-Controlled Repetition
- Formulating Algorithms: Sentinel-Controlled Repetition
- Formulating Algorithms: Nested Control Statements
- Compound Assignment Operators
- Increment and Decrement Operators
- Primitive Types

© 1992-2007 Pearson Education, Inc. All rights reserved.

Algorithms

- **Algorithms**
  - **The actions to execute**
  - **The order in which these actions execute**
- **Program control**
  - **Specifies the order in which actions execute in a program**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Pseudocode

- **Pseudocode**
  - **An informal language similar to English**
  - **Helps programmers develop algorithms**
  - **Does not run on computers**
  - **Should contain input, output and calculation actions**
  - **Should not contain variable declarations**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Control Structures

- Sequential execution
  - Statements are normally executed one after the other in the order in which they are written
- Transfer of control
  - Specifying the next statement to execute that is not necessarily the next one in order
  - Can be performed by the `goto` statement
    - Structured programming eliminated `goto` statements

© 1992-2007 Pearson Education, Inc. All rights reserved.

Control Structures (Cont.)

- Bohm and Jacopini's research
  - Demonstrated that `goto` statements were unnecessary
  - Demonstrated that all programs could be written with three control structures
    - The sequence structure,
    - The selection structure and
    - The repetition structure

© 1992-2007 Pearson Education, Inc. All rights reserved.

Control Structures (Cont.)

- UML activity diagram (www.uml.org)
  - Models the workflow (or activity) of a part of a software system
  - Action-state symbols (rectangles with their sides replaced with outward-curving arcs)
    - represent action expressions specifying actions to perform
  - Diamonds
    - Decision symbols
    - Merge symbols

© 1992-2007 Pearson Education, Inc. All rights reserved.

Control Structures (Cont.)

- Small circles
  - Solid circle represents the activity's initial state
  - Solid circle surrounded by a hollow circle represents the activity's final state
- Transition arrows
  - Indicate the order in which actions are performed
- Notes (rectangles with the upper-right corners folded over)
  - Explain the purposes of symbols (like comments in Java)
  - Are connected to the symbols they describe by dotted lines

© 1992-2007 Pearson Education, Inc. All rights reserved.

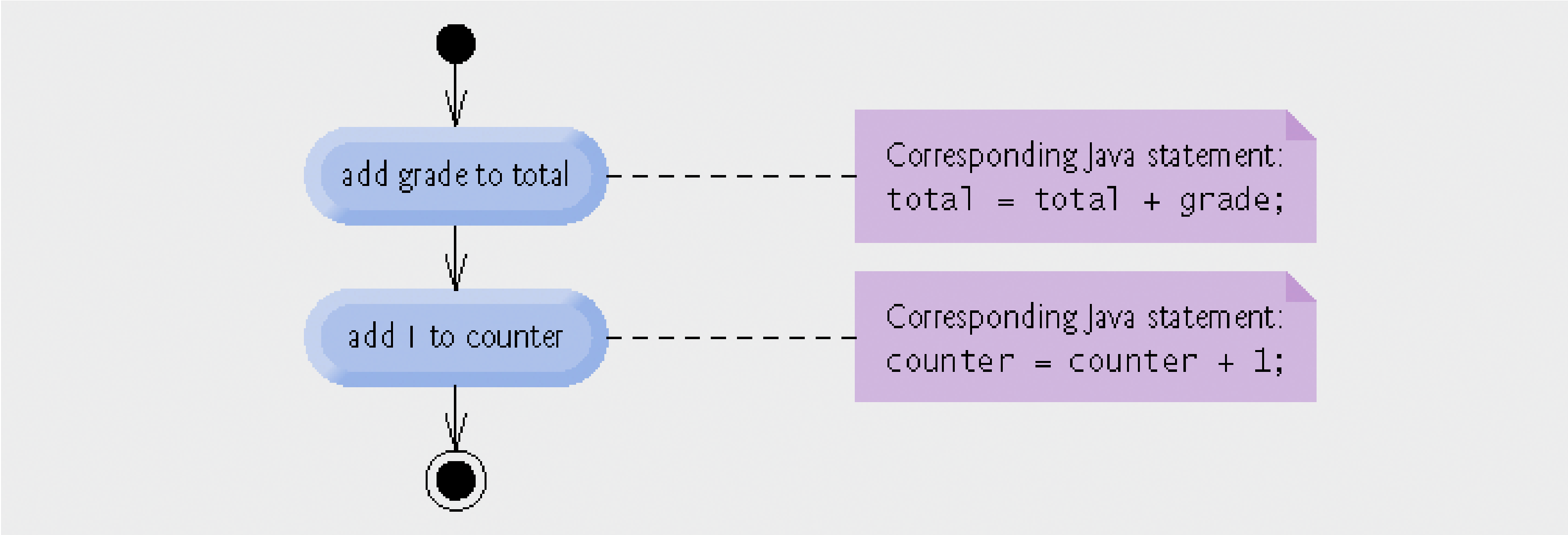


Fig. 4.1 | **Sequence structure activity diagram.**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Control Structures (Cont.)

- **Selection Statements**
  - **`if` statement**
    - **Single-selection statement**
  - **`if`...`else` statement**
    - **Double-selection statement**
  - **`switch` statement**
    - **Multiple-selection statement**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Control Structures (Cont.)

- Repetition statements
  - Also known as looping statements
  - Repeatedly performs an action while its loop-continuation condition remains true
  - `while` statement
    - Performs the actions in its body zero or more times
  - `do`…`while` statement
    - Performs the actions in its body one or more times
  - `for` statement
    - Performs the actions in its body zero or more times

© 1992-2007 Pearson Education, Inc. All rights reserved.

Control Structures (Cont.)

- Java has three kinds of control structures
  - Sequence statement,
  - Selection statements (three types) and
  - Repetition statements (three types)
  - All programs are composed of these control statements
    - Control-statement stacking
      - All control statements are single-entry/single-exit
    - Control-statement nesting

© 1992-2007 Pearson Education, Inc. All rights reserved.

`if` Single-Selection Statement

- `if` statements
  - Execute an action if the specified condition is `true`
  - Can be represented by a decision symbol (diamond) in a UML activity diagram
    - Transition arrows out of a decision symbol have guard conditions
      - Workflow follows the transition arrow whose guard condition is true

© 1992-2007 Pearson Education, Inc. All rights reserved.

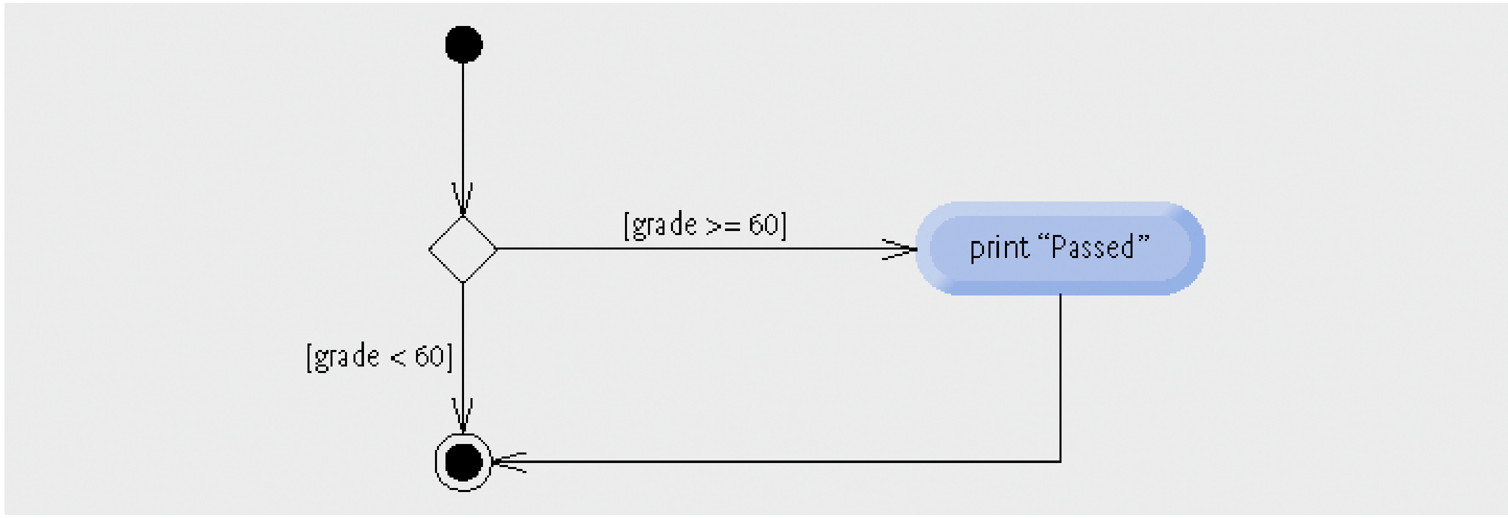


Fig. 4.2 | `if` single-selection statement UML activity diagram.

© 1992-2007 Pearson Education, Inc. All rights reserved.

`if…else` Double-Selection Statement

- **`if…else` statement**
  - Executes one action if the specified condition is `true` or a different action if the specified condition is `false`
- **Conditional Operator (`? :`)**
  - Java's only ternary operator (takes three operands)
  - `? :` and its three operands form a conditional expression
    - Entire conditional expression evaluates to the second operand if the first operand is `true`
    - Entire conditional expression evaluates to the third operand if the first operand is `false`

© 1992-2007 Pearson Education, Inc. All rights reserved.

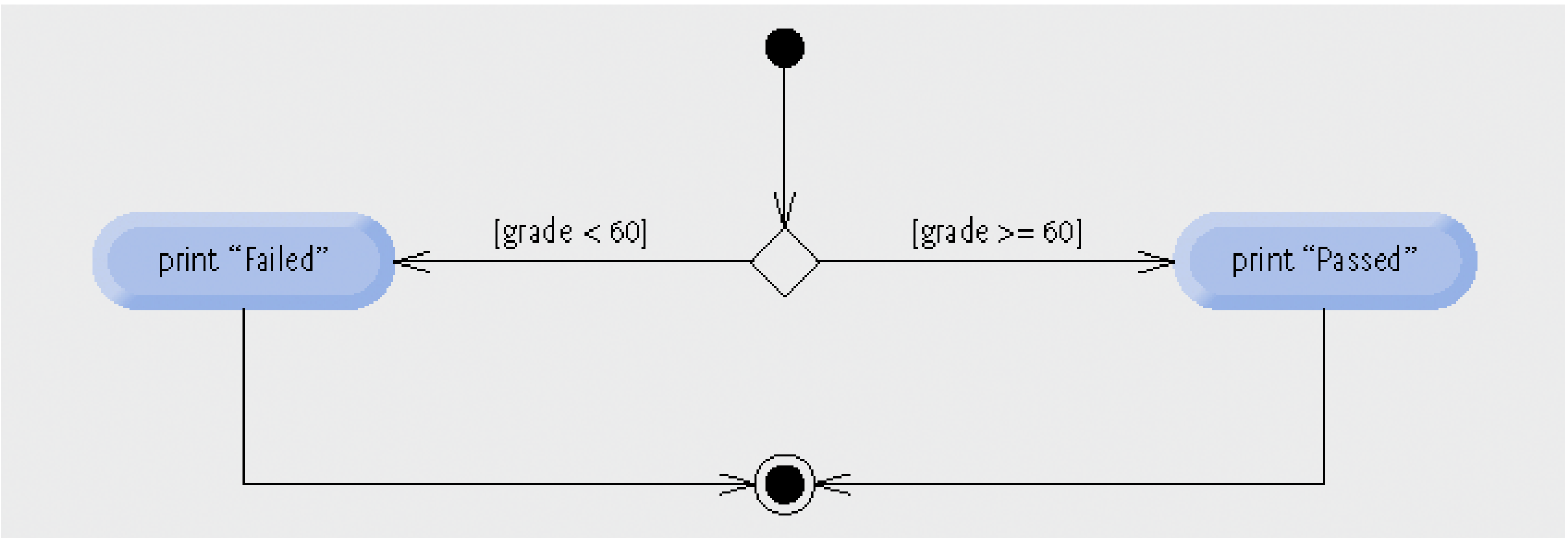


Fig. 4.3 | `if...else` double-selection statement UML activity diagram.

© 1992-2007 Pearson Education, Inc. All rights reserved.

`if…else` Double-Selection Statement (Cont.)

- **Nested `if…else` statements**
  - **`if…else` statements can be put inside other `if…else` statements**
- **Dangling-`else` problem**
  - **`else`s are always associated with the immediately preceding `if` unless otherwise specified by braces `{ }`**
- **Blocks**
  - **Braces `{ }` associate statements into blocks**
  - **Blocks can replace individual statements as an `if` body**

© 1992-2007 Pearson Education, Inc. All rights reserved.

`if…else` Double-Selection Statement (Cont.)

- **Logic errors**
  - **Fatal logic errors cause a program to fail and terminate prematurely**
  - **Nonfatal logic errors cause a program to produce incorrect results**
- **Empty statements**
  - **Represented by placing a semicolon (`;`) where a statement would normally be**
  - **Can be used as an `if` body**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Good Programming Practice 4.4

Always using braces in an `if...else` (or other) statement helps prevent their accidental omission, especially when adding statements to the `if`-part or the `else`-part at a later time. To avoid omitting one or both of the braces, some programmers type the beginning and ending braces of blocks before typing the individual statements within the braces.

© 1992-2007 Pearson Education, Inc. All rights reserved.

`while` Repetition Statement

- `while` statement
  - Repeats an action while its loop-continuation condition remains true
  - Uses a merge symbol in its UML activity diagram
    - Merges two or more workflows
    - Represented by a diamond (like decision symbols) but has:
      - Multiple incoming transition arrows,
      - Only one outgoing transition arrow and
      - No guard conditions on any transition arrows

© 1992-2007 Pearson Education, Inc. All rights reserved.

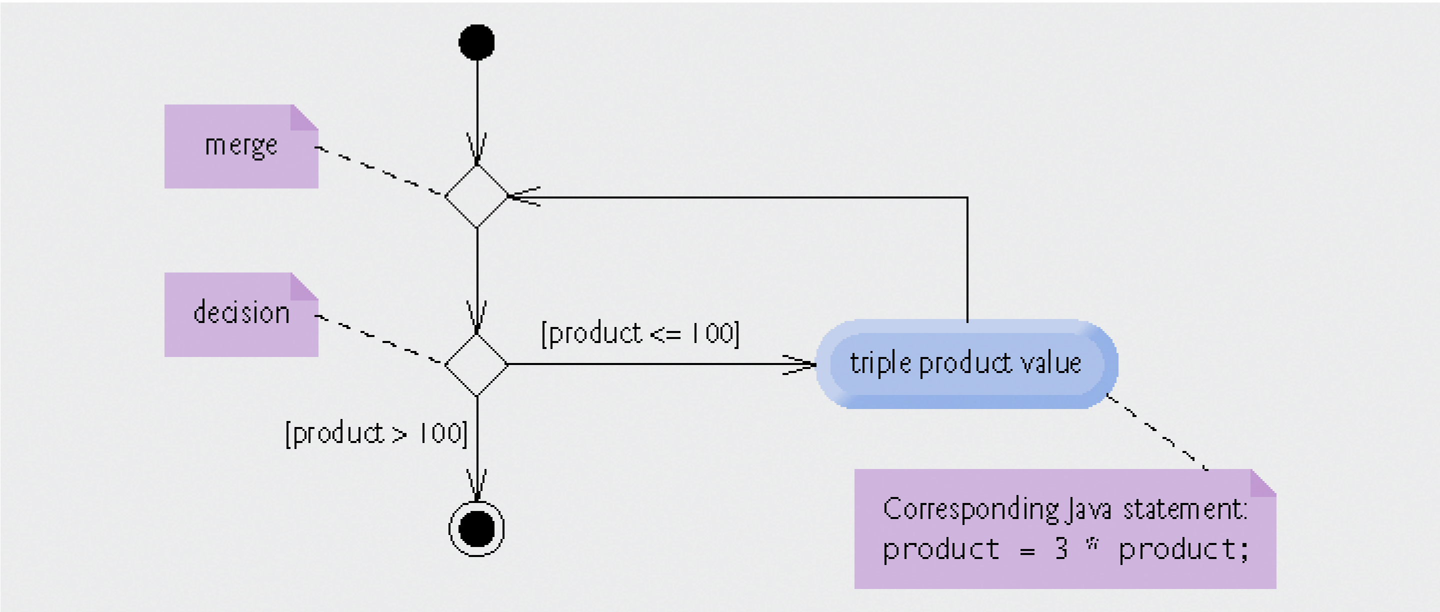


Fig. 4.4 | `while` repetition statement UML activity diagram.

© 1992-2007 Pearson Education, Inc. All rights reserved.

Formulating Algorithms: Counter-Controlled Repetition

- **Counter-controlled repetition**
  - **Use a counter variable to count the number of times a loop is iterated**
- **Integer division**
  - **The fractional part of an integer division calculation is truncated (thrown away)**

© 1992-2007 Pearson Education, Inc. All rights reserved.

```
1     Set total to zero
2     Set grade counter to one
3
4     While grade counter is less than or equal to ten
5         Prompt the user to enter the next grade
6         Input the next grade
7         Add the grade into the total
8         Add one to the grade counter
9
10    Set the class average to the total divided by ten
11    Print the class average
```

Fig. 4.5 | Pseudocode algorithm that uses counter-controlled repetition to solve the class-average problem.

© 1992-2007 Pearson Education, Inc. All rights reserved.

```java
import java.util.Scanner; // program uses class Scanner.
public class GradeBook .
{.
   public static void main(String[] args) .
   {.
      // create Scanner to obtain input from command window.
      Scanner input = new Scanner( System.in );.
.
      int total; // sum of grades entered by user.
      int gradeCounter; // number of the grade to be entered next.
      int grade; // grade value entered by user.
      int average; // average of grades.
.
      // initialization phase.
      total = 0; // initialize total.
      gradeCounter = 1; // initialize loop counter.
      while ( gradeCounter <= 10 ) // loop 10 times.
      {.
         System.out.print( "Enter grade: " ); // prompt .
         grade = input.nextInt(); // input next grade.
         total = total + grade; // add grade to total.
         gradeCounter = gradeCounter + 1; // increment counter by 1.
      } // end while.
   .
      // termination phase.
      average = total / 10; // integer division yields integer result.
.
      // display total and average of grades.
      System.out.printf( "\nTotal of all 10 grades is %d\n", total );.
      System.out.printf( "Class average is %d\n", average );.
   } // end method determineClassAverage.
} // end class GradeBook.
```

•GradeBook.java

›n, Inc. All rights reserved.

```
C:\Documents and Settings\Omer\Desktop\Java Course\Lecture1>java GradeBook
Enter grade: 12
Enter grade: 8
Enter grade: 12
Enter grade: 12
Enter grade: 3
Enter grade: 5
Enter grade: 6
Enter grade: 8
Enter grade: 9
Enter grade: 6

Total of all 10 grades is 81
Class average is 8

C:\Documents and Settings\Omer\Desktop\Java Course\Lecture1>
```

© 1992-2007 Pearson Education, Inc. All rights reserved.

Common Programming Error 4.5

Assuming that integer division rounds (rather than truncates) can lead to incorrect results. For example, 7 ÷ 4, which yields 1.75 in conventional arithmetic, truncates to 1 in integer arithmetic, rather than rounding to 2.

© 1992-2007 Pearson Education, Inc. All rights reserved.

Formulating Algorithms: Sentinel-Controlled Repetition

- **Sentinel-controlled repetition**
  - **Also known as indefinite repetition**
  - **Use a sentinel value (also known as a signal, dummy or flag value)**
    - **A sentinel value cannot also be a valid input value**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Common Programming Error 4.6

Choosing a sentinel value that is also a legitimate data value is a logic error.

© 1992-2007 Pearson Education, Inc. All rights reserved.

Error-Prevention Tip 4.2

When performing division by an expression whose value could be zero, explicitly test for this possibility and handle it appropriately in your program (e.g., by printing an error message) rather than allow the error to occur

© 1992-2007 Pearson Education, Inc. All rights reserved.

```
1     Initialize total to zero
2     Initialize counter to zero
3
4     Prompt the user to enter the first grade
5     Input the first grade (possibly the sentinel)
6
7     While the user has not yet entered the sentinel
8         Add this grade into the running total
9         Add one to the grade counter
10        Prompt the user to enter the next grade
11        Input the next grade (possibly the sentinel)
12
13    If the counter is not equal to zero
14        Set the average to the total divided by the counter
15        Print the average
16    else
17        Print “No grades were entered”
```

Fig. 4.8 | Class-average problem pseudocode algorithm with sentinel-controlled repetition.

© 1992-2007 Pearson Education, Inc. All rights reserved.

```
import java.util.Scanner; // program uses class Scanner.
public class GradeBookWhile .
{.
   public static void main(String[] args) .
   {.
      // create Scanner to obtain input from command window.
      Scanner input = new Scanner( System.in );.
.
      int total; // sum of grades entered by user.
      int gradeCounter; // number of the grade to be entered next.
      int grade; // grade value entered by user.
      double average; // average of grades.
.
      // initialization phase.
      total = 0; // initialize total.
      gradeCounter = 0; // initialize loop counter.
      .
      System.out.print( "Enter grade or -1 to quit: " ); // prompt .
      grade = input.nextInt(); // input next grade.
         .
      while ( grade != -1 ) {.
         total = total + grade; // add grade to total.
         gradeCounter = gradeCounter + 1; // increment counter by 1.
         System.out.print( "Enter grade or -1 to quit: " ); // prompt
         grade = input.nextInt(); // input next grade.
      } // end while.
   .
      // termination phase.
      average = (double) total / gradeCounter; .
      System.out.printf( "\nTotal of all 10 grades is %d\n", total );.
      System.out.printf( "Class average is %.2f\n", average );.
   } // end method determineClassAverage.
} // end class GradeBook.
```

ights reserved.

```
C:\Documents and Settings\Omer\Desktop\Java Course\Lecture1>java GradeBookWhile
Enter grade or -1 to quit: 34
Enter grade or -1 to quit: 16
Enter grade or -1 to quit: 5
Enter grade or -1 to quit: -1

Total of all 10 grades is 55
Class average is 18.33

C:\Documents and Settings\Omer\Desktop\Java Course\Lecture1>
```

© 1992-2007 Pearson Education, Inc. All rights reserved.

Formulating Algorithms: Sentinel-Controlled Repetition (Cont.)

- Unary cast operator
  - Creates a temporary copy of its operand with a different data type
    - example: `(double)` will create a temporary floating-point copy of its operand
  - Explicit conversion
- Promotion
  - Converting a value (e.g. `int`) to another data type (e.g. `double`) to perform a calculation
  - Implicit conversion

© 1992-2007 Pearson Education, Inc. All rights reserved.

Formulating Algorithms: Nested Control Statements

- **Control statements can be nested within one another**
  - **Place one control statement inside the body of the other**

© 1992-2007 Pearson Education, Inc. All rights reserved.

```
1  Initialize passes to zero
2  Initialize failures to zero
3  Initialize student counter to one
4
5  While student counter is less than or equal to 10
6     Prompt the user to enter the next exam result
7     Input the next exam result
8
9     If the student passed
10       Add one to passes
11    Else
12       Add one to failures
13
14    Add one to student counter
15
16 Print the number of passes
17 Print the number of failures
18
19 If more than eight students passed
20    Print “Raise tuition”
```

Fig. 4.11 | Pseudocode for examination-results problem.

© 1992-2007 Pearson Education, Inc. All rights reserved.

```
import java.util.Scanner; // class uses class Scanner.
public class Analysis .
{.
   public static void main(String[] args) .
   {.
      // create Scanner to obtain input from command window.
      Scanner input = new Scanner( System.in );.
      // initializing variables in declarations .
      int passes = 0; // number of passes         .
      int failures = 0; // number of failures     .
      int studentCounter = 1; // student counter.
      int result; // one exam result (obtains value from user).
      // process 10 students using counter-controlled loop.
      while ( studentCounter <= 10 ) .
      {.
         // prompt user for input and obtain value from user.
         System.out.print( "Enter result (1 = pass, 2 = fail): " );.
         result = input.nextInt();.
         // if...else nested in while                          .
         if ( result == 1 )           // if result 1,          .
            passes = passes + 1;      // increment passes;     .
         else                         // else result is not 1, so.
            failures = failures + 1; // increment failures     .
.
         // increment studentCounter so loop eventually terminates.
         studentCounter = studentCounter + 1;  .
      } // end while.
      System.out.printf( "Passed: %d\nFailed: %d\n", passes, failures );
      // determine whether more than 8 students passed.
      if ( passes > 8 ).
         System.out.println( "Hardworking class." );.
   } // end method.
} // end class Analysis.
```

hts reserved.

```
C:\Documents and Settings\Omer\Desktop\Java Course\Lecture1>java Analysis
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 9
Failed: 1
Hardworking class.

C:\Documents and Settings\Omer\Desktop\Java Course\Lecture1>
```

© 1992-2007 Pearson Education, Inc. All rights reserved.

Compound Assignment Operators

- **Compound assignment operators**
  - **An assignment statement of the form:**
    variable = variable operator expression ;
    where *operator* is `+`, `-`, `*`, `/` or `%` can be written as:
    variable operator= expression ;
  - **example: `c = c + 3;` can be written as `c += 3;`**
    - **This statement adds `3` to the value in variable `c` and stores the result in variable `c`**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`C = c + 7`	`10` to `c`
`-=`	`d -= 4`	`d = d - 4`	`1` to `d`
`*=`	`e *= 5`	`e = e * 5`	`20` to `e`
`/=`	`f /= 3`	`f = f / 3`	`2` to `f`
`%=`	`g %= 9`	`g = g % 9`	`3` to `g`

Fig. 4.14 | Arithmetic compound assignment operators.

© 1992-2007 Pearson Education, Inc. All rights reserved.

Increment and Decrement Operators

- Unary increment and decrement operators
  - Unary increment operator (++) adds one to its operand
  - Unary decrement operator (−−) subtracts one from its operand
  - Prefix increment (and decrement) operator
    - Changes the value of its operand, then uses the new value of the operand in the expression in which the operation appears
  - Postfix increment (and decrement) operator
    - Uses the current value of its operand in the expression in which the operation appears, then changes the value of the operand

© 1992-2007 Pearson Education, Inc. All rights reserved.

Operator	Called	Sample expression	Explanation
`++`	prefix increment	`++a`	Increment `a` by `1`, then use the new value of `a` in the expression in which `a` resides.
`++`	postfix increment	`a++`	Use the current value of `a` in the expression in which `a` resides, then increment `a` by `1`.
`--`	prefix decrement	`--b`	Decrement `b` by `1`, then use the new value of `b` in the expression in which `b` resides.
`--`	postfix decrement	`b--`	Use the current value of `b` in the expression in which `b` resides, then decrement `b` by `1`.

Fig. 4.15 | Increment and decrement operators.

© 1992-2007 Pearson Education, Inc. All rights reserved.

```java
1  // Fig. 4.16: Increment.java
2  // Prefix increment and postfix increment operators.
3
4  public class Increment
5  {
6     public static void main( String args[] )
7     {
8        int c;
9
10       // demonstrate postfix increment operator
11       c = 5; // assign 5 to c
12       System.out.println( c );   // print 5
13       System.out.println( c++ ); // print 5 then postincrement
14       System.out.println( c );   // print 6
15
16       System.out.println(); // skip a line
17
18       // demonstrate prefix increment operator
19       c = 5; // assign 5 to c
20       System.out.println( c );   // print 5
21       System.out.println( ++c ); // preincrement then print 6
22       System.out.println( c );   // print 6
23
24    } // end main
25
26 } // end class Increment
```

Postincrementing the `c` variable

Preincrementing the `c` variable

```
5
5
6

5
6
6
```

- Increment.java

© 1992-2007 Pearson Education, Inc. All rights reserved.

Operators						Associativity	Type
++	--					right to left	unary postfix
++	--	+	-	(*type*)		right to left	unary prefix
*	/	%				left to right	Multiplicative
+	-					left to right	Additive
<	<=	>	>=			left to right	Relational
==	!=					left to right	Equality
?:						right to left	Conditional
=	+=	-=	*=	/=	%=	right to left	assignment

Fig. 4.17 | Precedence and associativity of the operators discussed so far.

© 1992-2007 Pearson Education, Inc. All rights reserved.

Primitive Types

- **Java is a strongly typed language**
  - **All variables have a type**
- **Primitive types in Java are portable across all platforms that support Java**

© 1992-2007 Pearson Education, Inc. All rights reserved.

Portability Tip 4.1

Unlike C and C++, the primitive types in Java are portable across all computer platforms that support Java. Thanks to this and Java's many other portability features, a programmer can write a program once and be certain that it will execute on any computer platform that supports Java. This capability is sometimes referred to as *WORA* (Write Once, Run Anywhere).

© 1992-2007 Pearson Education, Inc. All rights reserved.