Course Information

- Six Lectures
- Teaches “Java Standard 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

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
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

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.)

Java Platform

Java Standart Edition (SE)
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)

Java Micro Edition (ME)

- geared toward applications for small, memory constrained devices
  - Midlets
    - Google Maps Mobile
    - Opera Mini

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

Use Java API classes

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

Fig. 1.1 | Typical Java development environment.

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

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
First Program in Java: Printing a Line of Text

```
// Fig. 2.1: Welcome1.java
public class Welcome1
{
   // main method begins execution of Java application
   public static void main(String args[])
   {
      System.out.println("Welcome to Java Programming!");
   }
   // end method main
}
// end class Welcome1

Welcome to Java Programming!
```

- 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 */

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

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

- 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

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

4 public class welcome {

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

5 {

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

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 ;

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)

11 } // end method main

- Ends method declaration

13 } // end class welcome

- Ends class declaration
- Can add comments to keep track of ending braces
• 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

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

Modifying Our First Java Program

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

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.
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

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 Java Programming!");
```
- Line breaks at `\n`
### Displaying Text with printf

- **System.out.printf**
  - Feature added in Java SE 5.0
  - Displays formatted data
    ```java
    9         System.out.printf("%s\n\n", "Welcome to", "Java Programming! ");
    10
    ```
  - Format string
    - Fixed text
    - Format specifier – placeholder for a value
    - Format specifier `%s` – placeholder for a string

---

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

<table>
<thead>
<tr>
<th>Escape sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\n</code></td>
<td>Newline. Position the screen cursor at the beginning of the next line.</td>
</tr>
<tr>
<td><code>\t</code></td>
<td>Horizontal tab. Move the screen cursor to the next tab stop.</td>
</tr>
<tr>
<td><code>\r</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>\ </code></td>
<td>Backslash. Used to print a backslash character.</td>
</tr>
<tr>
<td><code>&quot;\&quot;</code></td>
<td>Double quote. Used to print a double-quote character. For example, System.out.println(&quot;&quot;&quot;in quotes&quot;&quot;&quot;&quot;); displays &quot;in quotes&quot;</td>
</tr>
</tbody>
</table>

---

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

Declares and initializes variables number1, number2 and sum.

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

Another Java Application: Adding Integers (Cont.)

```
3 import java.util.Scanner; // program uses class Scanner
   // 1. 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
   // 2. Begins public class Addition
     • Recall that file name must be Addition.java
     • Lines 8-9: begin main
```

Outline

1. Addition.java
   (1 of 2)
   // main method begins execution of Java application
   public static void main(String[] args)
   {
     // create scanner to obtain input from command window
     Scanner input = new Scanner(System.in);
     // read first number from user
     System.out.println("Enter first integer: ");
     number1 = input.nextInt();
     // read second number from user
     System.out.println("Enter second integer: ");
     number2 = input.nextInt();
     // add numbers
     sum = number1 + number2;
     // display the sum using formatted output
     System.out.printf("sum is \%.1f\n", sum);
   }
   // end main
   // end class Addition
```

Outline

2. Addition.java
   (1 of 2)
   // program uses class Scanner

3. Addition.java
   // program uses class Scanner

4. Addition.java
   // create scanner to obtain input from command window

5. Addition.java
   // create scanner to obtain input from command window

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

© 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, s, \n, 7
     • int, float, double and char are primitive types
   -- Can add comments to describe purpose of variables
16  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
```

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()
```

Another Java Application: Adding Integers (Cont.)

```
20  System.out.print("Enter second integers "); // 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
```

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

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

![Fig. 2.8 | Memory location showing the name and value of variable number1.](image)

![Fig. 2.9 | Memory locations after storing values for number1 and number2.](image)

![Fig. 2.10 | Memory locations after calculating and storing the sum of number1 and number2.](image)
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

<table>
<thead>
<tr>
<th>Java operation</th>
<th>Arithmetic operator</th>
<th>Algebraic expression</th>
<th>Java expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>f + 7</td>
<td>f + 7</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>p - c</td>
<td>p - c</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>b * m</td>
<td></td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>x / y or ( \frac{x}{y} ) or x + y</td>
<td>x / y</td>
</tr>
</tbody>
</table>

Fig. 2.11 | Arithmetic operators.

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

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

Fig. 2.12 | Precedence of arithmetic operators.
### 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)

---

#### Standard algebraic equality or relational operator

<table>
<thead>
<tr>
<th>Java equality or relational operator</th>
<th>Sample Java condition</th>
<th>Meaning of Java condition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Equality operators</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>==</td>
<td>x == y</td>
<td>x is equal to y</td>
</tr>
<tr>
<td>!=</td>
<td>x != y</td>
<td>x is not equal to y</td>
</tr>
<tr>
<td><strong>Relational operators</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>x &gt; y</td>
<td>x is greater than y</td>
</tr>
<tr>
<td>&lt;</td>
<td>x &lt; y</td>
<td>x is less than y</td>
</tr>
<tr>
<td>&gt;=</td>
<td>x &gt;= y</td>
<td>x is greater than or equal to y</td>
</tr>
<tr>
<td>&lt;=</td>
<td>x &lt;= y</td>
<td>x is less than or equal to y</td>
</tr>
</tbody>
</table>

---

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

---

### Outline

#### Comparison.java

(1 of 2)

1. Class Comparison
   1.1 Main
   1.2 Declarations
   1.3 Import data (nextInt)
   1.4 Compare two inputs using `if` statements

#### Test for equality, display result using `printf`.

#### Compares two numbers using relational operator `<`.

---

**Fig. 2.14** | Equality and relational operators.
Decision Making: Equality and Relational Operators (Cont.)

Outline

Comparison.java
(2 of 2)

Program output

Decision Making: Equality and Relational Operators (Cont.)

- 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

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

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

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*

Object-oriented Programming

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

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
Object-oriented Programming

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

Object-oriented Programming

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

Control Statements
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

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

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

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
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

- 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

![Sequence structure activity diagram.](image)

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

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

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

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

**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` is 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

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

**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
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.

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

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)

Fig. 4.4 while repetition statement UML activity diagram.
Fig. 4.5 | Pseudocode algorithm that uses counter-controlled repetition to solve the class-average problem.

```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 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.
        System.out.printf("Total of all 10 grades is %d\n", total);
        System.out.printf("Class average is %d\n", average);
    } // end method determineClassAverage.
} // end class GradeBook.
```

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.
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

Common Programming Error 4.6

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

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

```
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.
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

---

Formulating Algorithms: Nested Control Statements

- Control statements can be nested within one another
  - Place one control statement inside the body of the other
Fig. 4.11 | Pseudocode for examination-results problem.

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
Passed: 9
Failed: 1
Hardworking class.

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

---

**Fig. 4.14** | Arithmetic compound assignment operators.

<table>
<thead>
<tr>
<th>Assignment operator</th>
<th>Sample expression</th>
<th>Explanation</th>
<th>Assigns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+=</code></td>
<td><code>c += 7</code></td>
<td><code>c = c + 7</code></td>
<td><code>10 to c</code></td>
</tr>
<tr>
<td><code>-=</code></td>
<td><code>d -= 4</code></td>
<td><code>d = d - 4</code></td>
<td><code>1 to d</code></td>
</tr>
<tr>
<td><code>*=</code></td>
<td><code>e *= 5</code></td>
<td><code>e = e * 5</code></td>
<td><code>20 to e</code></td>
</tr>
<tr>
<td><code>/=</code></td>
<td><code>f /= 3</code></td>
<td><code>f = f / 3</code></td>
<td><code>2 to f</code></td>
</tr>
<tr>
<td><code>%=</code></td>
<td><code>g %= 9</code></td>
<td><code>g = g % 9</code></td>
<td><code>3 to g</code></td>
</tr>
</tbody>
</table>

---

**Fig. 4.15** | Increment and decrement operators.

```java
// Fig. 4.16: Increment.java
// Prefix increment and postfix increment operators,
public class Increment
{
    public static void main(String[] args)
    {
        int c;
        // demonstrate postfix increment operator
        c = 5; // assign 5 to c
        System.out.println(c); // print 5
        System.out.println(c++); // print 5 then postincrement
        System.out.println(c); // print 6
        System.out.println(); // skip a line
        // demonstrate prefix increment operator
        c = 5; // assign 5 to c
        System.out.println(c); // print 5
        System.out.println(c++); // preincrement then print 5
        System.out.println(c); // print 6
    }
}
```

---

© 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**

<table>
<thead>
<tr>
<th>Operators</th>
<th>Associativity</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>++ --</td>
<td>right to left</td>
<td>unary postfix</td>
</tr>
<tr>
<td>+ -</td>
<td>left to right</td>
<td>Additive</td>
</tr>
<tr>
<td>&gt; &lt; &gt;= &lt;=</td>
<td>= left to right</td>
<td>Relational</td>
</tr>
<tr>
<td>? :</td>
<td>right to left</td>
<td>Conditional</td>
</tr>
<tr>
<td>- / %</td>
<td>right to left</td>
<td>Unary prefix</td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
<td>Multiplicative</td>
</tr>
<tr>
<td>&lt;= &gt;=</td>
<td>left to right</td>
<td>Equality</td>
</tr>
<tr>
<td>= !=</td>
<td>right to left</td>
<td>Assignment</td>
</tr>
</tbody>
</table>

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

---

**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).