Introduction to Java Applications
2.2 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
// Fig. 2.1: Welcome1.java

// Text-printing program.

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

// end class Welcome1

Welcome to Java Programming!
2.2 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
2.2 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

beginning class declaration for 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
2.2 First Program in Java: Printing a Line of Text (Cont.)

- **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
- **In chapters 2 to 7, use `public class`**
  - Certain details not important now
  - Mimic certain features, discussions later

```java
public class Welcome1
```

© 2005 Pearson Education, Inc. All rights reserved.
2.2 First Program in Java: Printing a Line of Text (Cont.)

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

- **Left brace {**
  - Begins body of every class
  - Right brace ends declarations (line 13)
2.2 First Program in Java: Printing a Line of Text (Cont.)

- Part of every Java application
  - Applications begin executing at `main`
    - Parentheses indicate `main` is a method (Ch. 3 and 6)
    - 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

- Left brace begins body of method declaration
  - Ended by right brace `}` (line 11)
Indent the entire body of each method declaration one “level” of indentation between the left brace, {, and the right brace, }, that define the body of the method. This format makes the structure of the method stand out and makes the method declaration easier to read.
Instructs computer to perform an action
  • Prints string of characters
    – String - series 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 ;
2.2 First Program in Java: Printing a Line of Text (Cont.)

- Ends method declaration

- Ends class declaration
- Can add comments to keep track of ending braces
2.2 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
2.2 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`
2.3 Modifying Our First Java Program

• Modify example in Fig. 2.1 to print same contents using different code
2.3 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
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

Welcome to Java Programming!
2.3 Modifying Our First Java Program (Cont.)

• **Escape characters**
  – Backslash (\)
  – Indicates special characters 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
Welcome3.java

1. main

public static void main( String args[] )
{
    System.out.println( "Welcome\nto\nJava\nProgramming!" );
}

Program Output

Welcome
to
Java
Programming!

Notice how a new line is output for each \n escape sequence.
2.4 Displaying Text with printf

- **System.out.printf**
  - New feature of J2SE 5.0
  - Displays formatted data

```java
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
public class Welcome4 {

    public static void main( String args[] ) {

        System.out.printf( "%s
%s
" , "Welcome to", "Java Programming!" );

    } // end method main

} // end class Welcome4

Welcome to
Java Programming!
2.5 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
// Fig. 2.7: Addition.java
// Addition program that displays the sum of two numbers.
import java.util.Scanner; // program uses class Scanner

public class Addition
{
    // 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 );

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

        System.out.print( "Enter first integer: " ); // prompt
        number1 = input.nextInt(); // read first number from user
System.out.print("Enter second integer: "); // prompt

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

sum = number1 + number2; // add numbers

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

} // end method main

} // end class Addition

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

Enter 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.

Two integers entered by the user.
import declarations

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

public class Addition
{

- Begins public class Addition
  - Recall that file name must be Addition.java
- Lines 8-9: begins main
Error-Prevention Tip 2.7

Forgetting to include an `import` declaration for a class used in your program typically results in a compilation error containing a message such as “cannot resolve symbol.” When this occurs, check that you provided the proper `import` declarations and that the names in the `import` declarations are spelled correctly, including proper use of uppercase and lowercase letters.
2.5 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`
2.5 Another Java Application: Adding Integers (Cont.)

- 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

```
13 int number1; // first number to add
14 int number2; // second number to add
15 int sum; // second number to add
```

- Can declare multiple variables of the same type in one declaration
- Use comma-separated list
Good Programming Practice 2.11

Choosing meaningful variable names helps a program to be *self-documenting* (i.e., one can understand the program simply by reading it rather than by reading manuals or viewing an excessive number of comments).
Good Programming Practice 2.12

By convention, variable-name identifiers begin with a lowercase letter, and every word in the name after the first word begins with a capital letter. For example, variable-name identifier `firstNumber` has a capital `N` in its second word, `Number`. 
2.5 Another Java Application: Adding Integers (Cont.)

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

```java
17 System.out.print( "Enter first integer: " ); // prompt
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()`
Software Engineering Observation 2.1

By default, package `java.lang` is imported in every Java program; thus, `java.lang` is the only package in the Java API that does not require an import declaration.
2.5 Another Java Application: Adding Integers (Cont.)

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

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

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

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

- 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

```java
sum = number1 + number2; // add numbers
```
2.5 Another Java Application: Adding Integers (Cont.)

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

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

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
2.7 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
### Precedence of arithmetic operators

<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.*
Using parentheses for complex arithmetic expressions, even when the parentheses are not necessary, can make the arithmetic expressions easier to read.
2.8 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>=</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>&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. 2.14** | Equality and relational operators.
// Fig. 2.15: Comparison.java
// Compare integers using if statements, relational operators
// and equality operators.
import java.util.Scanner; // program uses class Scanner

public class Comparison
{
    // 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 );

        int number1; // first number to compare
        int number2; // second number to compare

        System.out.print( "Enter first integer: " ); // prompt
        number1 = input.nextInt(); // read first number from user

        System.out.print( "Enter second integer: " ); // prompt
        number2 = input.nextInt(); // read second number from user

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

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

        if ( number1 < number2 )
            System.out.printf( "%d < %d\n", number1, number2 );
    }
}
if ( number1 > number2 )
    System.out.printf( "%d > %d\n", number1, number2 );

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

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

} // end method main

} // end class Comparison

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
2.8 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
2.8 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 if statement
  - 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

```java
23  if ( number1 == number2 )
24    System.out.printf( "%d == %d\n", number1, number2 );
```
Common Programming Error 2.10

Confusing the equality operator, ==, with the assignment operator, =, can cause a logic error or a syntax error. The equality operator should be read as “is equal to,” and the assignment operator should be read as “gets” or “gets the value of.” To avoid confusion, some people read the equality operator as “double equals” or “equals equals.”
Indent an if statement’s body to make it stand out and to enhance program readability.
Common Programming Error 2.13

Placing a semicolon immediately after the right parenthesis of the condition in an if statement is normally a logic error.
2.9 (Optional) Software Engineering Case Study: Examining the Requirements Document

• **Object-oriented design (OOD) process using UML**
  – Chapters 3 to 8, 10

• **Object-oriented programming (OOP) implementation**
  – Appendix J
• Requirements Document
  – New automated teller machine (ATM)
  – Allows basic financial transaction
    • View balance, withdraw cash, deposit funds
  – User interface
    • Display screen, keypad, cash dispenser, deposit slot
  – ATM session
    • Authenticate user, execute financial transaction
Fig. 2.17 | Automated teller machine user interface.
Fig. 2.18 | ATM main menu.
Fig. 2.19 | ATM withdrawal menu.
2.9 (Optional) Software Engineering Case Study (Cont.)

• Analyzing the ATM System
  – Requirements gathering
  – Software life cycle
    • Waterfall model
    • Interactive model
  – Use case modeling

• Use case Diagram
  – Model the interactions between clients and its use cases
  – Actor
    • External entity
Fig. 2.20 | Use case diagram for the ATM system from the user's perspective.
Fig. 2.21 | Use case diagram for a modified version of our ATM system that also allows users to transfer money between accounts.
• UML diagram types
  – Model system structure
• Class diagram
  – Models classes, or “building blocks” of a system
  – screen, keypad, cash dispenser, deposit slot.
2.9 (Optional) Software Engineering Case Study (Cont.)

- Model system behavior
  - Use case diagrams
    - Model interactions between user and a system
  - State machine diagrams
    - Model the ways in which an object changes state
  - Activity diagrams
    - Models an object’s activity during program execution
  - Communication diagrams (collaboration diagrams)
    - Models the interactions among objects
    - Emphasize what interactions occur
  - Sequence diagrams
    - Models the interactions among objects
    - Emphasize when interactions occur