Introduction to Java Concepts

Introduction

- Over the years, many programmers learned structured programming.
- You'll learn structured programming and object-oriented programming—the key programming methodology used by programmers today.
- You'll create and work with many software objects.
  - Their internal structure is often built using structured-programming techniques.
- The logic of manipulating objects is occasionally expressed with structured programming.

Introduction (Cont.)

- Java has become the language of choice for implementing Internet-based applications and software for devices that communicate over a network.
- There are now billions of Java-enabled mobile phones and handheld devices.
- Java is the preferred language for meeting many organizations' enterprisewide programming needs.

Machine Languages, Assembly Languages and High-Level Languages

- Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate translation steps.
- Three general language types:
  - Machine languages
  - Assembly languages
  - High-level languages

Machine Languages, Assembly Languages and High-Level Languages (Cont.)

- Any computer can directly understand only its own machine language.
  - This is the computer's "natural language," defined by its hard-ware de-sign.
  - Generally consist of strings of numbers (ultimately reduced to 1s and 0s) that instruct computers to perform their most elementary operations one at a time.
  - Machine dependent—a particular ma-chine language can be used on only one type of computer.

Machine Languages, Assembly Languages and High-Level Languages (Cont.)

- Englishlike abbreviations that represent elementary operations formed the basis of assembly languages.
- Translator programs called assemblers convert assembly-language programs to machine language.

Machine Languages, Assembly Languages and High-Level Languages (Cont.)

- High-level languages
  - Single statements accomplish substantial tasks.
  - Compilers convert high-level language programs into machine language.
  - Allow you to write instructions that look almost like everyday English and contain commonly used mathematical notations.
- C, C++, Microsoft's .NET languages (e.g., Visual Basic, Visual C++ and C#) are among the most widely used high-level programming languages; Java is by far the most widely used.

Machine Languages, Assembly Languages and High-Level Languages (Cont.)

- Compiling a high-level language program into machine language can take a considerable amount of computer time.
- Interpreter programs execute high-level language programs directly, although slower than compiled programs run.
- Java uses a clever mixture of compilation and interpretation to run programs.

Java Class Libraries

- Java programs consist of pieces called classes.
- Classes include methods that perform tasks and return information when the tasks complete.
- Java class libraries
  - Rich collections of existing classes
  - Also known as the Java APIs (Application Programming Interfaces)
- Two aspects to learning the Java "world."
  - The Java language it-self
  - The classes in the extensive Java class libraries
- Download the Java API documentation
  - `java.sun.com/javase/downloads/`
  - Scroll down to the Additional Resources section and click the Download button to the right of Java SE 6 Documentation.

Typical Java Development Environment

- Java program development and execution cycle.

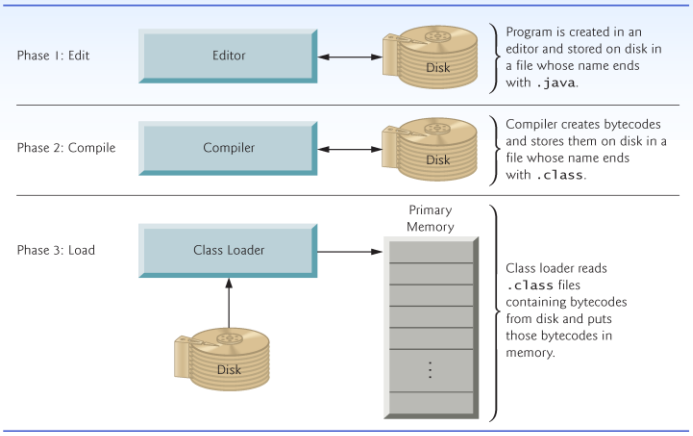

Fig. 1.1 | Typical Java development environment. (Part 1 of 2.)

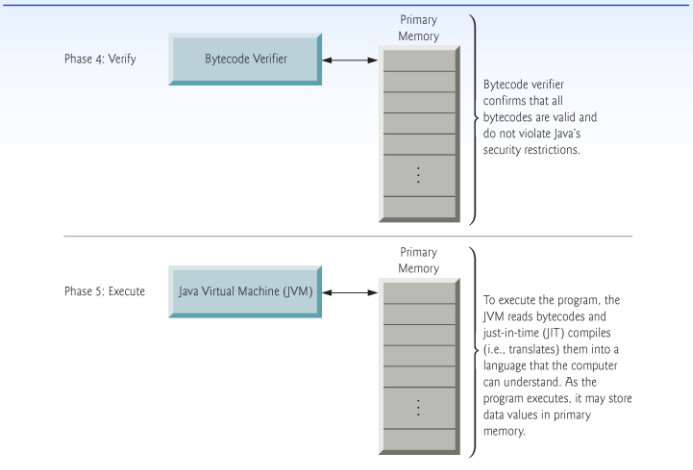

Fig. 1.1 | Typical Java development environment. (Part 2 of 2.)

Typical Java Development Environment (Cont.)

- Java programs normally go through five phases
  - edit
  - compile
  - load
  - verify
  - execute

Typical Java Development Environment (Cont.)

- Phase 1 consists of editing a file with an editor program *(normally known simply as an editor)*.
  - Type a Java program (source code) using the editor
  - Make any necessary corrections
  - Save the program
    - A file name ending with the `.java` extension indicates that the file contains Java source code.
  - Linux editors: `vi` and `emacs`.
  - Windows editors: Notepad, EditPlus (`www.editplus.com`), TextPad (`www.textpad.com`) and jEdit (`www.jedit.org`).

Typical Java Development Environment (Cont.)

- Integrated development environments (IDEs)
  - Provide tools that support the software-development process, including editors for writing and editing programs and debuggers for locating logic errors—errors that cause programs to execute incorrectly.
- Popular IDEs
  - Eclipse (`www.eclipse.org`)
  - NetBeans (`www.netbeans.org`)
  - JBuilder (`www.codegear.com`)
  - JCreator (`www.jcreator.com`)
  - BlueJ (`www.blueJ.org`)
  - jGRASP (`www.jgrasp.org`)

Typical Java Development Environment (Cont.)

- Phase 2
  - Use the command `javac` (the Java compiler) to compile a program. For example, to compile a program called `Welcome.java`, you'd type

    ```
    javac Welcome.java
    ```

  - If the program compiles, the compiler produces a .class file called `Welcome.class` that contains the compiled version of the program.

Typical Java Development Environment (Cont.)

- Java compiler translates Java source code into bytecodes that represent the tasks to execute.
- Bytecodes are executed by the Java Virtual Machine (JVM)—a part of the JDK and the foundation of the Java platform.
- Virtual machine (VM)—a software application that simulates a computer
  - Hides the underlying operating system and hardware from the programs that interact with it.
- If the same VM is implemented on many computer platforms, applications that it executes can be used on all those platforms.

Typical Java Development Environment (Cont.)

- Bytecodes are platform independent
  - They do not depend on a particular hardware platform.
- Bytecodes are portable
  - The same bytecodes can execute on any platform containing a JVM that understands the version of Java in which the bytecodes were compiled.
- The JVM is invoked by the `java` command. For example, to execute a Java application called `Welcome`, you'd type the command

```
java Welcome
```

Typical Java Development Environment (Cont.)

- Phase 3
  - The JVM places the program in memory to execute it
    - This is known as loading.
  - Class loader takes the `.class` files containing the program's bytecodes and transfers them to primary memory.
  - Also loads any of the `.class` files provided by Java that your program uses.
    - The `.class` files can be loaded from a disk on your system or over a network.

Typical Java Development Environment (Cont.)

- Phase 4
  - As the classes are loaded, the bytecode verifier examines their bytecodes
    - Ensures that they are valid and do not violate Java's security restrictions.
  - Java enforces strong security to make sure that Java programs arriving over the network do not damage your files or your system (as computer viruses and worms might).

Typical Java Development Environment (Cont.)

- Phase 5
  - The JVM executes the program's bytecodes.
  - JVM typically uses a combination of interpretation and just-in-time (JIT) compilation.
  - Analyzes the bytecodes as they are interpreted, searching for hot spots—parts of the bytecodes that execute frequently.
  - A just-in-time (JIT) compiler (the Java HotSpot compiler) translates the bytecodes into the underlying computer's machine language.
  - When the JVM encounters these compiled parts again, the faster machine-language code executes.
  - Java programs actually go through two compilation phases
    - One in which source code is translated into bytecodes (for portability across computer platforms)
    - A second in which, during execution, the bytecodes are translated into machine language for the actual computer on which the program executes.

Introduction to Classes and Objects

OBJECTIVES

In this Chapter you'll learn:

- To write simple Java applications.
- To use input and output statements.
- Java's primitive types.
- Basic memory concepts.
- To use arithmetic operators.
- The precedence of arithmetic operators.
- To write decision-making statements.
- To use relational and equality operators.

Introduction

- Java application programming
- Use tools from the JDK to compile and run programs.

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

- Javadoc comments
  - Delimited by `/**` and `*/`.
  - All text between the Javadoc comment delimiters is ignored by the compiler.
  - Enable you to embed program documentation directly in your programs.
  - The `javadoc` utility program reads Javadoc comments and uses them to prepare your program's documentation in HTML format.

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

- Blank lines and space characters
  - Make programs easier to read.
  - Blank lines, spaces and tabs are known as white space (or whitespace).
  - White space is ignored by the compiler.

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

- Class declaration

```
public class Welcome1
```

  - Every Java program consists of at least one class that you define.
  - `class` keyword introduces a class declaration and is immediately followed by the class name.
  - Keywords are reserved for use by Java and are always spelled with all lowercase letters.

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

- Class names
  - By convention, begin with a capital letter and capitalize the first letter of each word they include (e.g., `SampleClassName`).
  - A class name is an identifier—a series of characters consisting of letters, digits, underscores (_) and dollar signs (`$`) that does not begin with a digit and does not contain spaces.
  - Java is case sensitive—uppercase and lowercase letters are distinct—so `a1` and `A1` are different (but both valid) identifiers.

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

- Declaring the `main` Method

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

  - Starting point of every Java application.
  - Parentheses after the identifier `main` indicate that it's a program building block called a method.
  - Java class declarations normally contain one or more methods.
  - `main` must be defined as shown; otherwise, the JVM will not execute the application.
  - Methods perform tasks and can return information when they complete their tasks.
  - Keyword `void` indicates that this method will not return any information.

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

- Body of the method declaration
  - Enclosed in left and right braces.
- Statement

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

  - Instructs the computer to perform an action
    - Print the string of characters contained between the double quotation marks.
  - A string is sometimes called a character string or a string literal.
  - White-space characters in strings are not ignored by the compiler.
  - Strings cannot span multiple lines of code.

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

- `System.out` object
  - Standard output object.
  - Allows Java applications to display strings in the command window from which the Java application executes.
- `System.out.println` method
  - Displays (or prints) a line of text in the command window.
  - The string in the parentheses the argument to the method.
  - Positions the output cursor at the beginning of the next line in the command window.
- Most statements end with a semicolon.

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

- Compiling and Executing Your First Java Application
  - Open a command window and change to the directory where the program is stored.
  - Many operating systems use the command `cd` to change directories.
  - To compile the program, type
    ```
    javac Welcome1.java
    ```
  - If the program contains no syntax errors, preceding command creates a `.class` file (known as the class file) containing the platform-independent Java bytecodes that represent the application.
  - When we use the `java` command to execute the application on a given platform, these bytecodes will be translated by the JVM into instructions that are understood by the underlying operating system.

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

- To execute the program, type `java Welcome1`.
- Launches the JVM, which loads the `.class` file for class `Welcome1`.
- Note that the `.class` file-name extension is omitted from the preceding command; otherwise, the JVM will not execute the program.
- The JVM calls method `main` to execute the program.

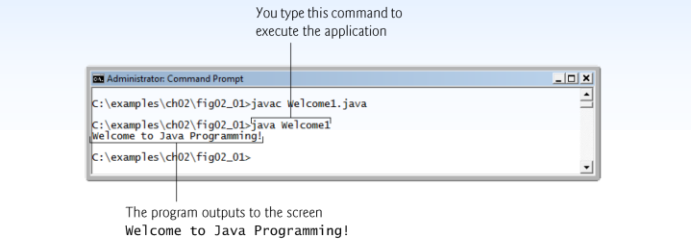

Fig. 2.2 | Executing `Welcome1` from the **Command Prompt**.

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.

Another Application: Adding Integers

- Integers
  - Whole numbers, like –22, 7, 0 and 1024)
- Programs remember numbers and other data in the computer's memory and access that data through program elements called variables.
- The program of Fig. 2.7 demonstrates these concepts.


```
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 a Scanner to obtain input from the 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
20       System.out.print( "Enter second integer: " ); // prompt
21       number2 = input.nextInt(); // read second number from user
22
23       sum = number1 + number2; // add numbers, then store total in sum
```

Imports class Scanner for use in this program

Creates Scanner for reading data from the user

Variables that are declared but not initialized

Reads an int value from the user

Reads another int value from the user

Sums the values of number1 and number2

Fig. 2.7 | Addition program that displays the sum of two numbers. (Part 1 of 2.)

```
24
25          System.out.printf( "Sum is %d\n", sum ); // display sum
26       } // end method main
27    } // end class Addition
```

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

Fig. 2.7 | Addition program that displays the sum of two numbers. (Part 2 of 2.)

Another Application: Adding Integers (Cont.)

- `import` declaration
  - Helps the compiler locate a class that is used in this program.
  - Rich set of predefined classes that you can reuse rather than "reinventing the wheel."
  - Classes are grouped into packages—named groups of related classes—and are collectively referred to as the Java class library, or the Java Application Programming Interface (Java API).
  - You use `import` declarations to identify the predefined classes used in a Java program.

Another Application: Adding Integers (Cont.)

- Variable declaration statement

```
Scanner input = new Scanner( System.in );
```

  - Specifies the name (`input`) and type (`Scanner`) of a variable that is used in this program.
- Variable
  - A location in the computer's memory where a value can be stored for use later in a program.
  - Must be declared with a name and a type before they can be used.
  - A variable's name enables the program to access the value of the variable in memory.
  - The name can be any valid identifier.
  - A variable's type specifies what kind of information is stored at that location in memory.

Another Application: Adding Integers (Cont.)

- `Scanner`
  - Enables a program to read data for use in a program.
  - Data can come from many sources, such as the user at the keyboard or a file on disk.
  - Before using a `Scanner`, you must create it and specify the source of the data.
- The equals sign (=) in a declaration indicates that the variable should be initialized (i.e., prepared for use in the program) with the result of the expression to the right of the equals sign.
- The `new` keyword creates an object.
- Standard input object, `System.in`, enables applications to read bytes of information typed by the user.
- `Scanner` object translates these bytes into types that can be used in a program.

Another Application: Adding Integers (Cont.)

- Variable declaration statements

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

  declare that variables `number1`, `number2` and `sum` hold data of type `int`
  - They can hold integer.
  - Range of values for an `int` is –2,147,483,648 to +2,147,483,647.
  - Actual `int` values may not contain commas.
- Several variables of the same type may be declared in one declaration with the variable names separated by commas.

Another Application: Adding Integers (Cont.)

- Prompt
  - Output statement that directs the user to take a specific action.
- `System` is a class.
  - Part of package `java.lang`.
  - Class `System` is not imported with an `import` declaration at the beginning of the program.

Another Application: Adding Integers (Cont.)

- `Scanner` method `nextInt`

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

  - Obtains an integer from the user at the keyboard.
  - Program waits for the user to type the number and press the Enter key to submit the number to the program.
- The result of the call to method `nextInt` is placed in variable `number1` by using the assignment operator, =.
  - "`number1` gets the value of `input.nextInt()`."
  - Operator = is called a binary operator—it has two operands.
  - Everything to the right of the assignment operator, =, is always evaluated before the assignment is performed.

Another Application: Adding Integers (Cont.)

- Arithmetic

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

  - Assignment statement that calculates the sum of the variables `number1` and `number2` then assigns the result to variable `sum` by using the assignment operator, =.
  - "`sum` gets the value of `number1` + `number2`."
  - In general, calculations are performed in assignment statements.
  - Portions of statements that contain calculations are called expressions.
  - An expression is any portion of a statement that has a value associated with it.

Another Application: Adding Integers (Cont.)

- Integer formatted output

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

  - Format specifier `%d` is a placeholder for an `int` value
  - The letter `d` stands for "decimal integer."

Memory Concepts

- Variables
  - Every variable has a name, a type, a size (in bytes) and a value.
  - When a new value is placed into a variable, the new value replaces the previous value (if any)
  - The previous value is lost.

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

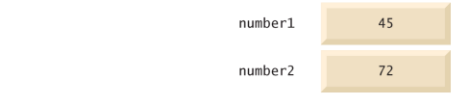

Fig. 2.9 | Memory locations after storing values for `number1` and `number2`.

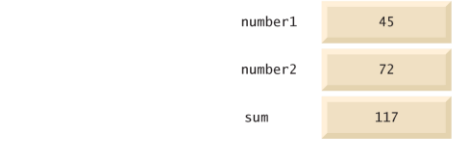

Fig. 2.10 | Memory locations after storing the sum of `number1` and `number2`.

Arithmetic

- Arithmetic operators are summarized in Fig. 2.11.
- The asterisk (*) indicates multiplication
- The percent sign (%) is the remainder operator
- The arithmetic operators are binary operators because they each operate on two operands.
- Integer division yields an integer quotient.
  - Any fractional part in integer division is simply discarded (i.e., truncated)—no rounding occurs.
- The remainder operator, %, yields the remainder after division.

Java operation	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`
Remainder	%	$r \bmod s$	`r % s`

Fig. 2.11 | Arithmetic operators.

Arithmetic (Cont.)

- Arithmetic expressions in Java must be written in straight-line form to facilitate entering programs into the computer.
- Expressions such as "`a` divided by `b`" must be written as `a / b`, so that all constants, variables and operators appear in a straight line.
- Parentheses are used to group terms in expressions in the same manner as in algebraic expressions.
- If an expression contains nested parentheses, the expression in the innermost set of parentheses is evaluated first.

Arithmetic (Cont.)

- Rules of operator precedence
  - Multiplication, division and remainder operations are applied first.
  - If an expression contains several such operations, they are applied from left to right.
  - Multiplication, division and remainder operators have the same level of precedence.
  - Addition and subtraction operations are applied next.
  - If an expression contains several such operations, the operators are applied from left to right.
  - Addition and subtraction operators have the same level of precedence.
- When we say that operators are applied from left to right, we are referring to their associativity.
- Some operators associate from right to left.
- Complete precedence chart is included in Appendix A.

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

Fig. 2.12 | Precedence of arithmetic operators.

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.


```
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
```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 1 of 3.)

```
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, number2 );
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      } // end method main
41   } // end class Comparison
```

Output statement executes only if the numbers are equal

Output statement executes only if the numbers are not equal

Output statement executes only if `number1` is less than `number2`

Output statement executes only if `number1` is greater than `number2`

Output statement executes only if `number1` is less than or equal to `number2`

Output statement executes only if `number1` is greater than or equal to `number2`

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 2 of 3.)


```
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
```

Fig. 2.15 | Compare integers using `if` statements, relational operators and equality operators. (Part 3 of 3.)