

# **Chapter 1 - Introduction to Java Applications**

# Basics of a Typical Java Environment

- Java programs normally undergo five phases
  - Edit
    - Programmer writes program (and stores program on disk)
  - Compile
    - Compiler creates *bytecodes* from program
  - Load
    - Class loader stores bytecodes in memory
  - Verify
    - Verifier ensures bytecodes do not violate security requirements
  - Execute
    - Interpreter translates bytecodes into machine language

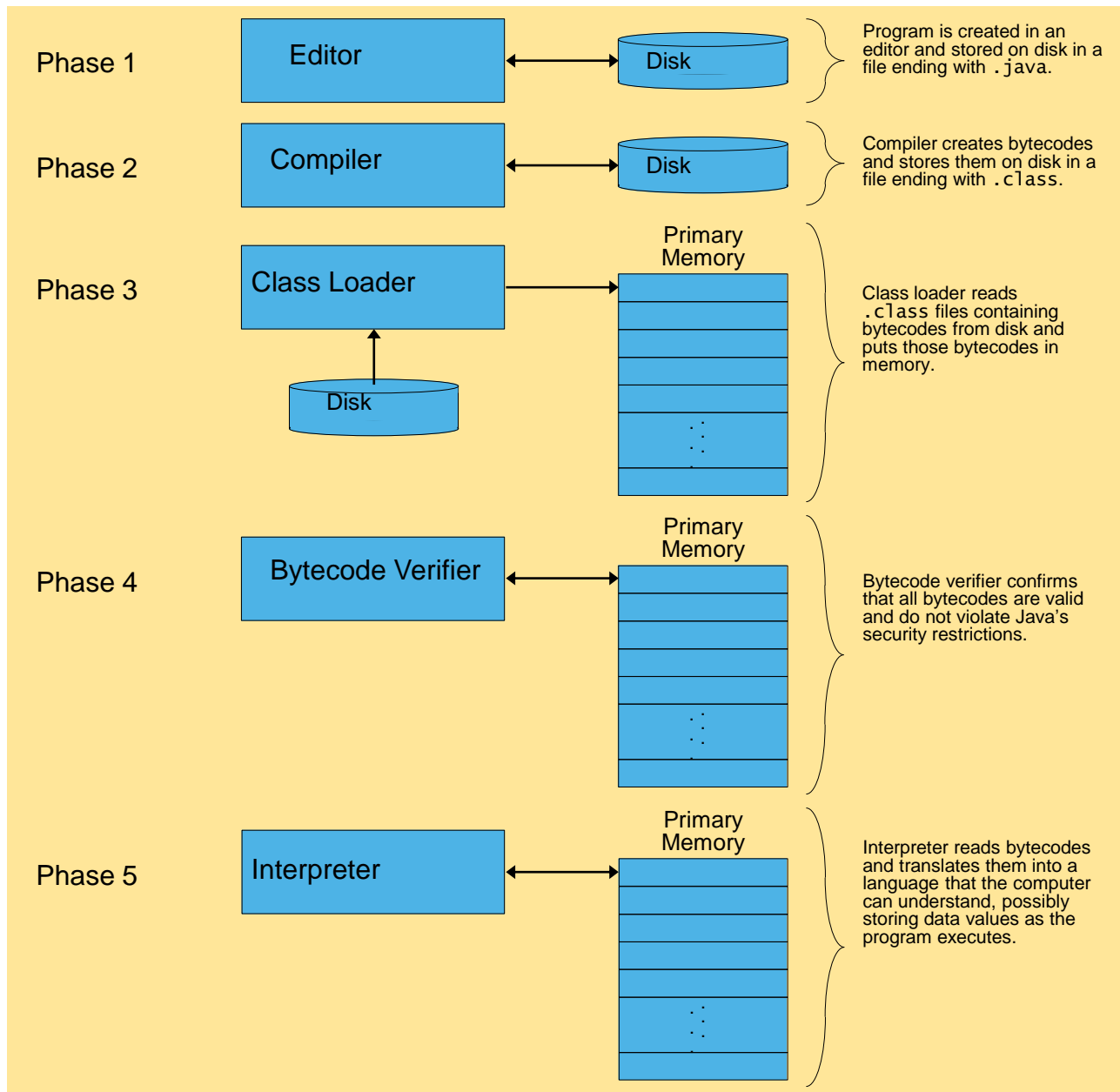


Fig. 1.1 Typical Java environment.

# A First Program in Java: Printing a Line of Text

```
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 class welcome1
```

welcome1.java

```
Welcome to Java Programming!
```

**Program Output**

# A First Program in Java: Printing a Line of Text

```
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

# A Simple Program: Printing a Line of Text

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`

# A Simple Program: Printing a Line of Text

```
4 public class welcome1 {
```

- Name of class called 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
- For chapters 2 to 7, use `public` keyword
  - Certain details not important now
  - Mimic certain features, discussions later

# A Simple Program: Printing a Line of Text

```
4 public class welcome1 {
```

- 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)

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

- Part of every Java application
  - Applications begin executing at `main`
    - Parenthesis indicate `main` is a method (ch. 6)
    - Java applications contain one or more methods



# A Simple Program: Printing a Line of Text

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

- 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)

# A Simple Program: Printing a Line of Text

9

```
system.out.println( "welcome to Java Programming!" );
```

- 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
  - Argument inside parenthesis
- This line known as a statement
  - Statements must end with semicolon ;

# A Simple Program: Printing a Line of Text

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

- Ends method declaration

```
13 } // end class welcome1
```

- Ends class declaration
- Can add comments to keep track of ending braces
- Lines 8 and 9 could be rewritten as:
- Remember, compiler ignores comments
- Comments can start on same line after code

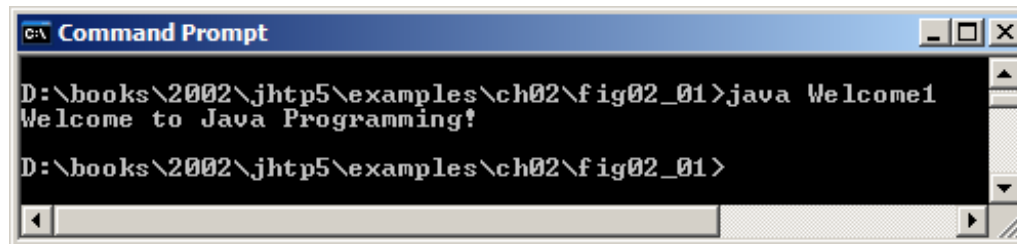
# A Simple Program: Printing a Line of Text

- Compiling a program
  - Open a command prompt window, go to directory where program is stored
  - Type `javac welcome1.java`
  - If no errors, `welcome1.class` created
    - Has bytecodes that represent application
    - Bytecodes passed to Java interpreter

# A Simple Program: Printing a Line of Text

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

Fig. 2.2 Executing Welcome1 in a Microsoft Windows 2000 Command Prompt.



```
C:\ Command Prompt
D:\books\2002\jhtp5\examples\ch02\fig02_01>java Welcome1
Welcome to Java Programming!
D:\books\2002\jhtp5\examples\ch02\fig02_01>
```

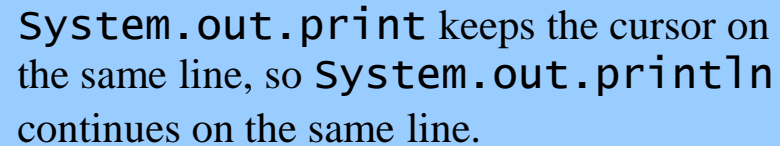
# Modifying Our First Java Program

- 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
```



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

welcome to Java Programming!

# Modifying Our First Java Program

- Newline characters (`\n`)
  - Interpreted as “special characters” by methods `System.out.print` and `System.out.println`
  - Indicates cursor should be on next line
  - `welcome3.java` (Fig. 2.4)

```
9      System.out.println( "welcome\n\tto\n\tJava\n\tProgramming!" );
```

- Line breaks at `\n`

- Usage
  - Can use in `System.out.println` or `System.out.print` to create new lines
    - `System.out.println( "welcome\n\tto\n\tJava\n\tProgramming!" );`



```
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!
```

Notice how a new line is output for each `\n` escape sequence.

# Modifying Our First Java Program

## Escape characters

- Backslash ( \ )
- Indicates special characters be output

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

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

# Displaying Text in a Dialog Box

- Display
  - Most Java applications use windows or a dialog box
    - We have used command window
  - Class `JOptionPane` allows us to use dialog boxes
- Packages
  - Set of predefined classes for us to use
  - Groups of related classes called *packages*
    - Group of all packages known as Java class library or Java applications programming interface (Java API)
  - `JOptionPane` is in the `javax.swing` package
    - Package has classes for using Graphical User Interfaces (GUIs)

```
1 // Fig. 2.6: welcome4.java
2 // Printing multiple lines in a dialog box.
3
4 // Java packages
5 import javax.swing.JOptionPane; // program uses JOptionPane
6
7 public class welcome4 {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12        JOptionPane.showMessageDialog(
13            null, "welcome\nto\nJava\nProgramming!" );
14
15        System.exit( 0 ); // terminate application with window
16
17    } // end method main
18
19 } // end class welcome4
```



**Program Output**

# Displaying Text in a Dialog Box

- Lines 1-2: comments as before

```
4 // Java packages
```

- Two groups of packages in Java API
- Core packages
  - Begin with `java`
  - Included with Java 2 Software Development Kit
- Extension packages
  - Begin with `javax`
  - New Java packages

```
5 import javax.swing.JOptionPane; // program uses JOptionPane
```

- `import` declarations
  - Used by compiler to identify and locate classes used in Java programs
  - Tells compiler to load class `JOptionPane` from `javax.swing` package

# Displaying Text in a Dialog Box

- Lines 6-11: Blank line, begin class `Welcome4` and `main`

```
12     JOptionPane.showMessageDialog(  
13         null, "Welcome\nto\nJava\nProgramming!" );
```

- Call method `showMessageDialog` of class `JOptionPane`
  - Requires two arguments
  - Multiple arguments separated by commas ( , )
  - For now, first argument always **null**
  - Second argument is string to display
- `showMessageDialog` is a `static` method of class `JOptionPane`
  - `static` methods called using class name, dot ( . ) then method name

# Displaying Text in a Dialog Box

- All statements end with ;
  - A single statement can span multiple lines
  - Cannot split statement in middle of identifier or string
- Executing lines 12 and 13 displays the dialog box



- Automatically includes an **OK** button
  - Hides or dismisses dialog box
- Title bar has string **Message**

# Displaying Text in a Dialog Box

```
15      System.exit( 0 ); // terminate application with window
```

- Calls `static` method `exit` of class `System`
  - Terminates application
    - Use with any application displaying a GUI
  - Because method is `static`, needs class name and dot (`.`)
  - Identifiers starting with capital letters usually class names
- Argument of `0` means application ended successfully
  - Non-zero usually means an error occurred
- Class `System` part of package `java.lang`
  - No `import` declaration needed
  - `java.lang` automatically imported in every Java program
- Lines 17-19: Braces to end `welcome4` and `main`



# Another Java Application: Adding Integers

- Upcoming program
  - Use input dialogs to input two values from user
  - Use message dialog to display sum of the two values

```

1 // Fig. 2.9: Addition.java
2 // Addition program that displays the sum of two numbers.
3
4 // Java packages
5 import javax.swing.JOptionPane; // program uses JOptionPane
6
7 public class Addition {
8
9 // main method begins execution
10 public static void main( String args[] )
11 {
12     String firstNumber; // first string entered by user
13     String secondNumber; // second string entered by user
14
15     int number1; // first integer entered by user
16     int number2; // second integer entered by user
17     int sum; // sum of two integers
18
19 // read in first number from user as a String
20 firstNumber = JOptionPane.showInputDialog( "Enter first integer" );
21
22 // read in second number from user as a String
23 secondNumber =
24     JOptionPane.showInputDialog( "Enter second integer" );
25
26 // convert numbers from type String to type int
27 number1 = Integer.parseInt( firstNumber );
28 number2 = Integer.parseInt( secondNumber );
29
30 // add numbers
31 sum = number1 + number2;
32

```

Declare variables: name and type.

Input first integer as a String, assign to firstNumber.

Convert strings to integers.

Add, place result in sum.

Addition.java

1. import

2. class Addition

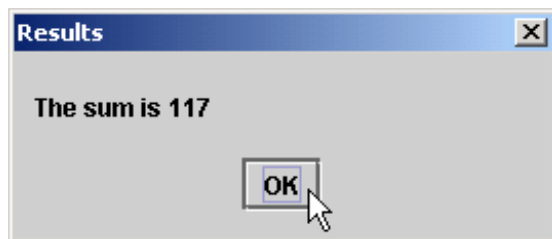
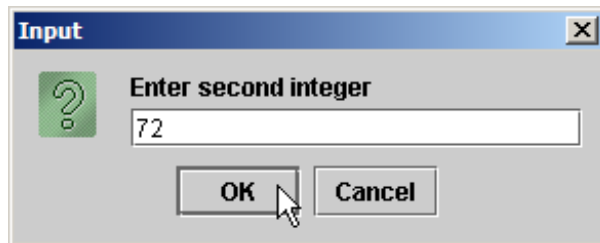
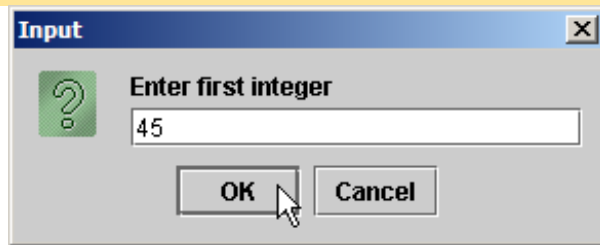
2.1 Declare variables  
(name and type)

3. showInputDialog

4. parseInt

5. Add numbers, put  
result in sum

```
33 // display result
34 JOptionPane.showMessageDialog( null, "The sum is " + sum,
35     "Results", JOptionPane.PLAIN_MESSAGE );
36
37     System.exit( 0 ); // terminate application with window
38
39 } // end method main
40
41 } // end class Addition
```



**Program output**

# Another Java Application: Adding Integers

```
5 import javax.swing.JOptionPane; // program uses JOptionPane
```

- Location of `JOptionPane` for use in the program

```
7 public class Addition {
```

- Begins `public class Addition`
  - Recall that file name must be `Addition.java`
- Lines 10-11: `main`

```
12     String firstNumber; // first string entered by user  
13     String secondNumber; // second string entered by user
```

- Declaration
  - `firstNumber` and `secondNumber` are variables

# Another Java Application: Adding Integers

```
12     String firstNumber; // first string entered by user
13     String secondNumber; // second string entered by user
```

## – Variables

- Location in memory that stores a value
  - Declare with name and type before use
- `firstNumber` and `secondNumber` are of type `String` (package `java.lang`)
  - Hold strings
- Variable name: any valid identifier
- Declarations end with semicolons ;

```
String firstNumber, secondNumber;
```

- Can declare multiple variables of the same type at a time
  - Use comma separated list
- Can add comments to describe purpose of variables

# Another Java Application: Adding Integers

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

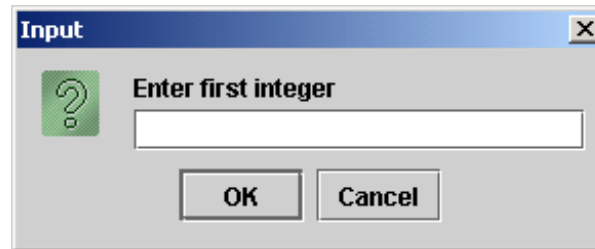
- Declares variables `number1`, `number2`, and `sum` of type `int`
  - `int` holds integer values (whole numbers): i.e., 0, -4, 97
  - Types `float` and `double` can hold decimal numbers
  - Type `char` can hold a single character: i.e., x, \$, \n, 7
  - Primitive types - more in Chapter 4

# Another Java Application: Adding Integers

20

```
firstNumber = JOptionPane.showInputDialog( "Enter first integer" );
```

- Reads **String** from the user, representing the first number to be added
  - Method `JOptionPane.showInputDialog` displays the following:



- Message called a prompt - directs user to perform an action
- Argument appears as prompt text
- If wrong type of data entered (non-integer) or click **Cancel**, error occurs

# Another Java Application: Adding Integers

20

```
firstNumber = JOptionPane.showInputDialog( "Enter first integer" );
```

- Result of call to `showInputDialog` given to `firstNumber` using assignment operator =
  - Assignment statement
  - = binary operator - takes two operands
    - Expression on right evaluated and assigned to variable on left
  - Read as: `firstNumber` gets value of `JOptionPane.showInputDialog( "Enter first integer" )`



# Another Java Application: Adding Integers

```
23     secondNumber =  
24         JOptionPane.showInputDialog( "Enter second integer" );
```

- Similar to previous statement
  - Assigns variable `secondNumber` to second integer input

```
27     number1 = Integer.parseInt( firstNumber );  
28     number2 = Integer.parseInt( secondNumber );
```

- Method `Integer.parseInt`
  - Converts `String` argument into an integer (type `int`)
    - Class `Integer` in `java.lang`
  - Integer returned by `Integer.parseInt` is assigned to variable `number1` (line 27)
    - Remember that `number1` was declared as type `int`
  - Line 28 similar

# Another Java Application: Adding Integers

31

```
sum = number1 + number2;
```

- 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

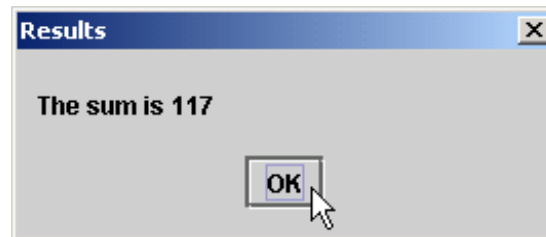
```
34 JOptionPane.showMessageDialog( null, "The sum is " + sum,  
35 "Results", JOptionPane.PLAIN_MESSAGE );
```

- Use `showMessageDialog` to display results
- "The sum is " + sum
  - Uses the operator `+` to "add" the string literal "The sum is" and `sum`
  - Concatenation of a `String` and another type
    - Results in a new string
  - If `sum` contains `117`, then "The sum is " + `sum` results in the new string "The sum is 117"
  - Note the space in "The sum is "
  - More on strings in Chapter 11





# Another Java Application: Adding Integers

```
34 JOptionPane.showMessageDialog( null, "The sum is " + sum,  
35 "Results", JOptionPane.PLAIN_MESSAGE );
```

- Different version of `showMessageDialog`
  - Requires four arguments (instead of two as before)
  - First argument: `null` for now
  - Second: string to display
  - Third: string in title bar
  - Fourth: type of message dialog with icon
    - Line 35 no icon: `JOptionPane.PLAIN_MESSAGE`



# Another Java Application: Adding Integers

Message dialog type	Icon	Description
<code>JOptionPane.ERROR_MESSAGE</code>		Displays a dialog that indicates an error to the user.
<code>JOptionPane.INFORMATION_MESSAGE</code>		Displays a dialog with an informational message to the user. The user can simply dismiss the dialog.
<code>JOptionPane.WARNING_MESSAGE</code>		Displays a dialog that warns the user of a potential problem.
<code>JOptionPane.QUESTION_MESSAGE</code>		Displays a dialog that poses a question to the user. This dialog normally requires a response, such as clicking on a <b>Yes</b> or a <b>No</b> button.
<code>JOptionPane.PLAIN_MESSAGE</code>	no icon	Displays a dialog that simply contains a message, with no icon.

**Fig. 2.12** `JOptionPane` constants for message dialogs.

# Arithmetic

- Arithmetic calculations used in most programs
  - Usage
    - \* for multiplication
    - / for division
    - +, -
  - Integer division truncates remainder
    - $7 / 5$  evaluates to 1
  - Remainder operator % returns the remainder
    - $7 \% 5$  evaluates to 2

# Arithmetic

- 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:  $( a + b + c ) / 3$
  - Follows **PEMDAS**
    - Parentheses, Exponents, Multiplication, Division, Addition, Subtraction

# Equality and Relational Operators

- `if` control statement
    - Simple version in this section, more detail later
    - If a condition is true, then the body of the `if` statement executed
      - `0` interpreted as false, non-zero is true
    - Control always resumes after the `if` structure
    - Conditions for `if` statements can be formed using equality or relational operators (next slide)
- ```
if ( condition )  
    statement executed if condition true
```
- No semicolon needed after condition
    - Else conditional task not performed



# Equality and Relational Operators

| Standard algebraic equality or relational operator | Java equality or relational operator | Example of Java condition | Meaning of Java condition       |
|----------------------------------------------------|--------------------------------------|---------------------------|---------------------------------|
| <i>Equality operators</i>                          |                                      |                           |                                 |
| =                                                  | ==                                   | x == y                    | x is equal to y                 |
|                                                    | !=                                   | x != y                    | x is not equal to y             |
| <i>Relational operators</i>                        |                                      |                           |                                 |
| >                                                  | >                                    | 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. 2.19** Equality and relational operators.

- Upcoming program uses `if` statements
  - Discussion afterwards

```
1 // Fig. 2.20: Comparison.java
2 // Compare integers using if statements, relational operators
3 // and equality operators.
4
5 // Java packages
6 import javax.swing.JOptionPane;
7
8 public class Comparison {
9
10 // main method begins execution of Java application
11 public static void main( String args[] )
12 {
13     String firstNumber; // first string entered by user
14     String secondNumber; // second string entered by user
15     String result; // a string containing the output
16
17     int number1; // first number to compare
18     int number2; // second number to compare
19
20 // read first number from user as a string
21 firstNumber = JOptionPane.showInputDialog( "Enter first integer:" );
22
23 // read second number from user as a string
24 secondNumber =
25     JOptionPane.showInputDialog( "Enter second integer:" );
26
27 // convert numbers from type String to type int
28 number1 = Integer.parseInt( firstNumber );
29 number2 = Integer.parseInt( secondNumber );
30
31 // initialize result to empty String
32 result = "";
33
```

Comparison.java

1. import

2. Class  
Comparison

2.1 main

2.2 Declarations

2.3 Input data  
(showInputDialog)

2.4 parseInt

2.5 Initialize result

```
34  if ( number1 == number2 )
35      result = result + number1 + " == " + number2;
36
37  if ( number1 != number2 )
38      result = result + number1 + " != " + number2;
39
40  if ( number1 < number2 )
41      result = result + "\n" + number1 + " < " + number2;
42
43  if ( number1 > number2 )
44      result = result + "\n" + number1 + " > " + number2;
45
46  if ( number1 <= number2 )
47      result = result + "\n" + number1 + " <= " + number2;
48
49  if ( number1 >= number2 )
50      result = result + "\n" + number1 + " >= " + number2;
51
52  // Display results
53  JOptionPane.showMessageDialog( null, result, "Comparison Results",
54      JOptionPane.INFORMATION_MESSAGE );
55
56  System.exit( 0 ); // terminate application
57
58  } // end method main
59
60  } // end class Comparison
```

Test for equality, create new string, assign to `result`.

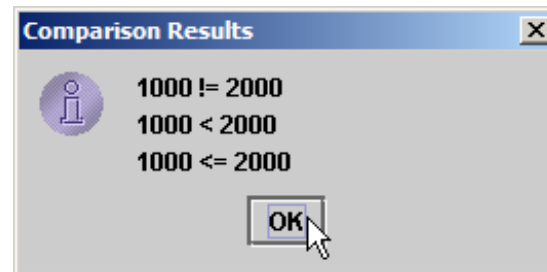
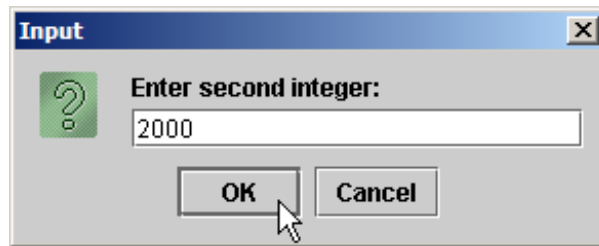
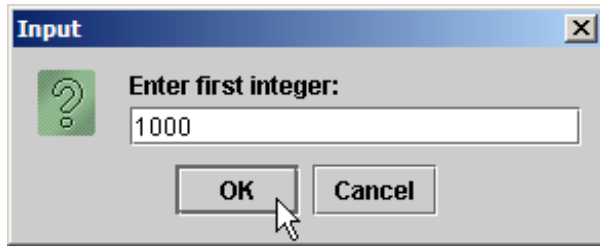
.java

### 3. if statements

### 4. showMessageDialog

Notice use of  
`JOptionPane.INFORMATION_MESSAGE`

## Program Output



# Equality and Relational Operators

- Precedence of operators
  - All operators except for = (assignment) associates from left to right
    - For example:  $x = y = z$  is evaluated  $x = (y = z)$

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

# Features of Java

- Simple
  - Architecture-neutral
  - Object-Oriented
  - Distributed
  - Compiled
  - Interpreted
  - Statically Typed
  - Multi-Threaded
  - Garbage Collected
- Portable
  - High-Performance
  - Robust
  - Secure
  - Extensible
  - Well-Understood

# How Will Java Change My Life?

- Get started quickly
- Write less code
- Write better code
- Develop programs faster
- Avoid platform dependencies with 100% pure Java
- Write once, run anywhere
- Distribute software more easily

# Java Applications and Java ... lets

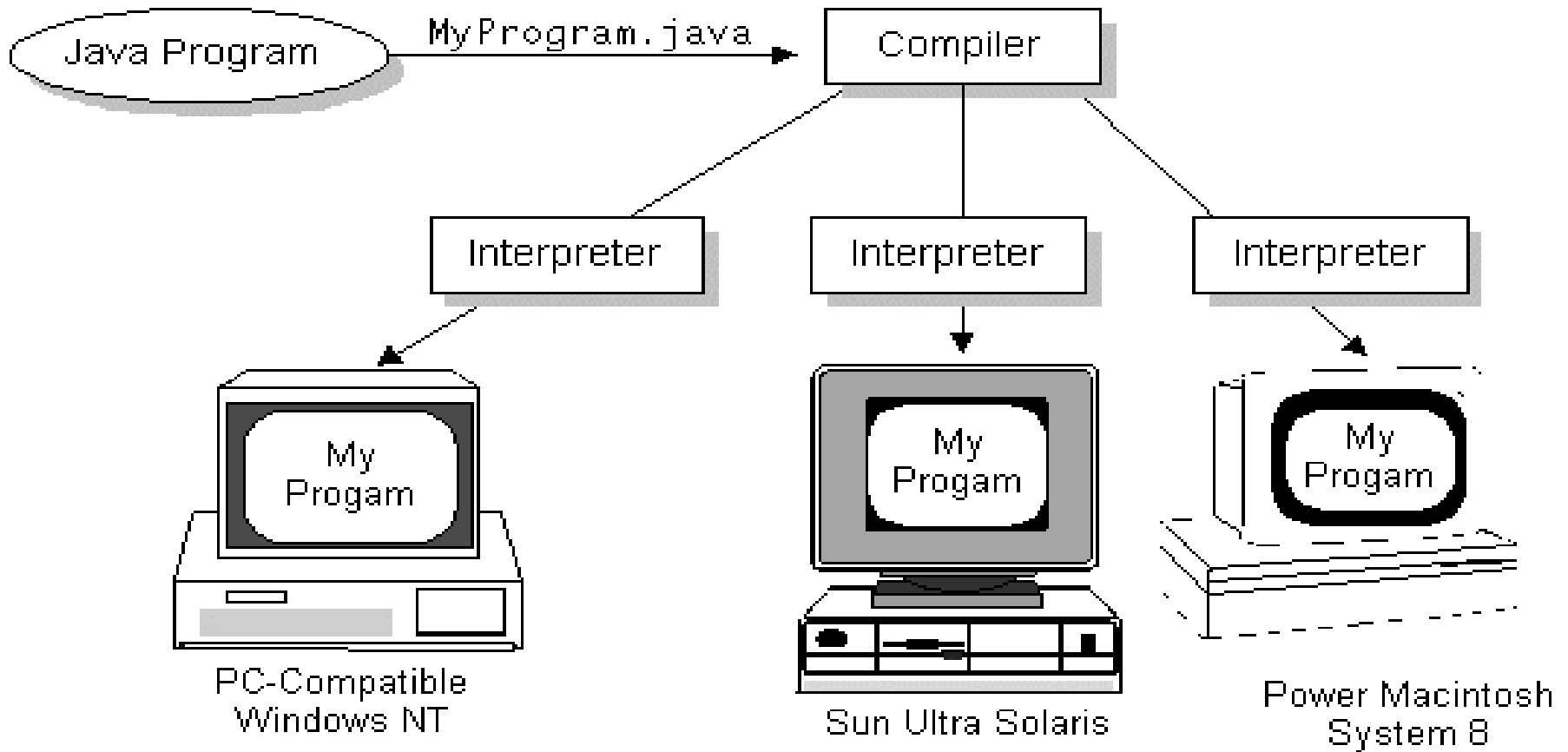
- Stand-alone Applications
  - Just like any programming language
- Applet
  - Run under a Java-Enabled Browser
- Midlet
  - Run in a Java-Enabled Mobile Phone
- Servlet
  - Run on a Java-Enabled Web Server



# Java Developer's Kit (I)

- Java's programming environment
  - Core Java API
  - compiler
  - interpreter
  - debugger
  - dis-assembler
  - profiler
  - more...

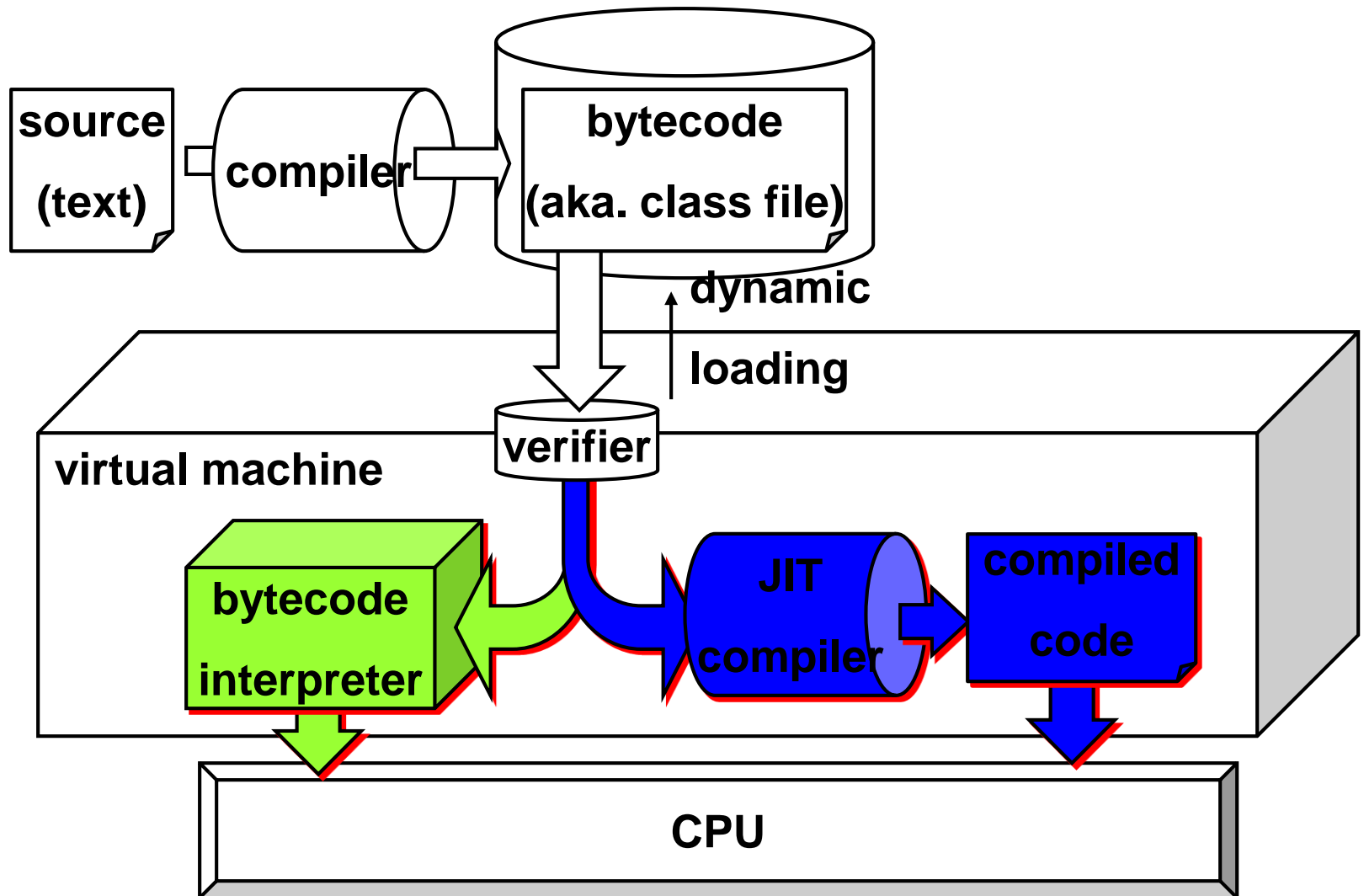
# Write Once, Run Anywhere



# ByteCode: Food for the VM

- For most languages, compilation produces machine code
- Java compilation produces “bytecode”
  - Intermediate code readable by the VM
  - Transferable across the Internet as *applets*
- VM interprets BC into instructions
- ByteCode produced on any platform may be executed on any other platform which supports a VM

# execution model of Java



# The JIT

- *Just-In-Time* compiler
- Translates bytecode into machine code at runtime
  - Performance increase 10-30 times
- Now the default for most JVM's
  - Can be turned off if desired
  - JIT can apply statistical optimizations based on runtime usage profile