Chapter 2 Elementary Programming

Motivations

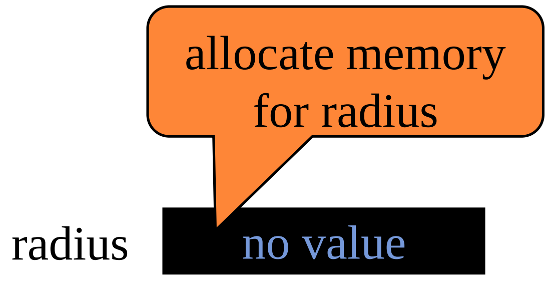
In the preceding chapter, you learned how to create, compile, and run a Java program. Starting from this chapter, you will learn how to solve practical problems programmatically. Through these problems, you will learn Java primitive data types and related subjects, such as variables, constants, data types, operators, expressions, and input and output.

Objectives

- To write Java programs to perform simple computations (§2.2).
- To obtain input from the console using the **Scanner** class (§2.3).
- To use identifiers to name variables, constants, methods, and classes (§2.4).
- To use variables to store data (§§2.5–2.6).
- To program with assignment statements and assignment expressions (§2.6).
- To use constants to store permanent data (§2.7).
- To name classes, methods, variables, and constants by following their naming conventions (§2.8).
- To explore Java numeric primitive data types: **byte**, **short**, **int**, **long**, **float**, and **double** (§2.9.1).
- To perform operations using operators **+**, **-**, *****, **/**, and **%** (§2.9.2).
- To perform exponent operations using **Math.pow(a, b)** (§2.9.3).
- To write integer literals, floating-point literals, and literals in scientific notation (§2.10).
- To write and evaluate numeric expressions (§2.11).
- To obtain the current system time using **System.currentTimeMillis()** (§2.12).
- To use augmented assignment operators (§2.13).
- To distinguish between postincrement and preincrement and between postdecrement and predecrement (§2.14).
- To cast the value of one type to another type (§2.15).
- To describe the software development process and apply it to develop the loan payment program (§2.16).
- To represent characters using the **char** type (§2.17).
- To represent a string using the **String** type (§2.18).
- To obtain input using the **JOptionPane** input dialog boxes (§2.19).

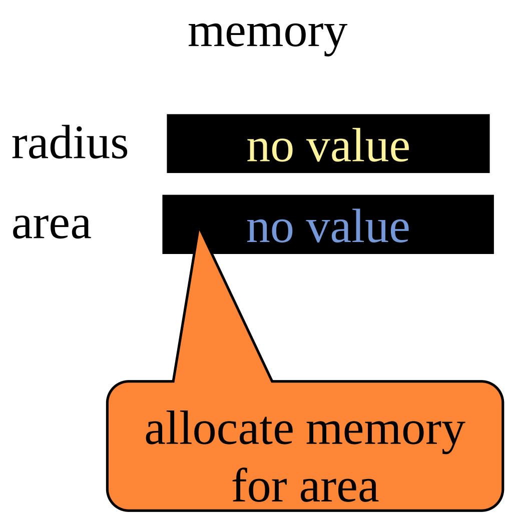
Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of
    radius " +
      radius + " is " + area);
  }
}
```



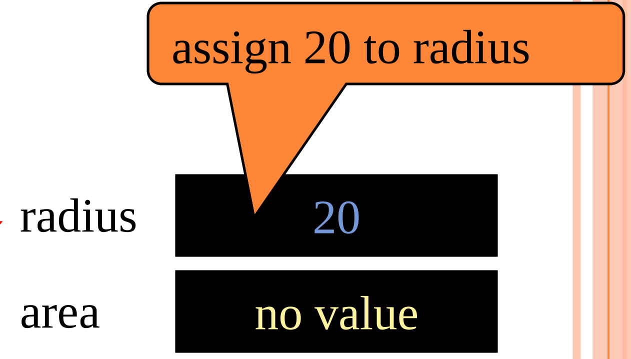
Trace a Program Execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



Trace a Program Execution

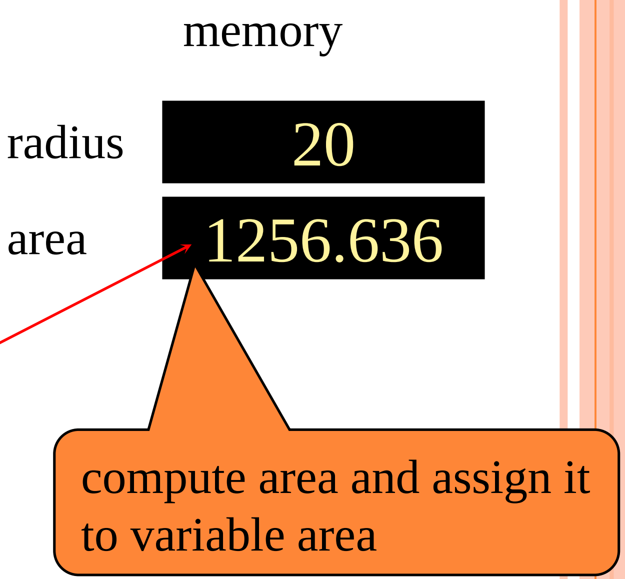
```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of 
    radius " +
      radius + " is " + area);
  }
}
```



animation

Trace a Program Execution

```
public class ComputeArea {
 /** Main method */
 public static void main(String[] args) {
   double radius;
   double area;

   // Assign a radius
   radius = 20;

   // Compute area
   area = radius * radius * 3.14159;

   // Display results
   System.out.println("The area for the circle of radius " +
     radius + " is " + area);
 }
}
```



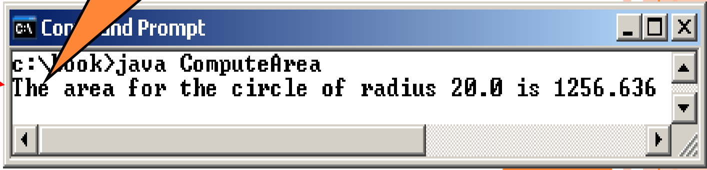
TRACE A PROGRAM EXECUTION

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```

memory

radius	20
area	1256.636

print a message to the console



```
c:\book>java ComputeArea
The area for the circle of radius 20.0 is 1256.636
```

IDENTIFIERS

- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).
- An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
- An identifier cannot be a reserved word.
- An identifier cannot be `true`, `false`, or `null`.
- An identifier can be of any length.

Declaring Variables

```
int x;         // Declare x to be an
               // integer variable;

double radius; // Declare radius to
               // be a double variable;

char a;        // Declare a to be a
               // character variable;
```

Assignment Statements

```
x = 1;               // Assign 1 to x;

radius = 1.0;        // Assign 1.0 to radius;

a = 'A';             // Assign 'A' to a;
```

Declaring and Initializing in One Step

- `int x = 1;`
- `double d = 1.4;`

NAMED CONSTANTS

```
final datatype CONSTANTNAME = VALUE;

final double PI = 3.14159;
final int SIZE = 3;
```

Naming Conventions

- Choose meaningful and descriptive names.
- Variables and method names:
  - Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name. For example, the variables `radius` and `area`, and the method `computeArea`.

Naming Conventions, cont.

- Class names:
  - Capitalize the first letter of each word in the name. For example, the class name `ComputeArea`.
- Constants:
  - Capitalize all letters in constants, and use underscores to connect words. For example, the constant `PI` `and` MAX_VALUE

Numerical Data Types

Name	Range	Storage Size
`byte`	-2^{7} to $2^{7}-1$ (-128 to 127)	8-bit signed
`short`	-2^{15} to $2^{15}-1$ (-32768 to 32767)	16-bit signed
`int`	-2^{31} to $2^{31}-1$ (-2147483648 to 2147483647)	32-bit signed
`long`	-2^{63} to $2^{63}-1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
`float`	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
`double`	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

NUMERIC OPERATORS

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

Integer Division

+, -, *, /, and %

5 / 2 yields an integer 2.

5.0 / 2 yields a double value 2.5

5 % 2 yields 1 (the remainder of the division)

NOTE

Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy. For example,

System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);

displays 0.5000000000000001, not 0.5, and

System.out.println(1.0 - 0.9);

displays 0.09999999999999998, not 0.1. Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.

Exponent Operations

```
System.out.println(Math.pow(2, 3));
// Displays 8.0
System.out.println(Math.pow(4, 0.5));
// Displays 2.0
System.out.println(Math.pow(2.5, 2));
// Displays 6.25
System.out.println(Math.pow(2.5, -2));
// Displays 0.16
```

Number Literals

A *literal* is a constant value that appears directly in the program. For example, 34, 1,000,000, and 5.0 are literals in the following statements:

```
int i = 34;
long x = 1000000;
double d = 5.0;
```

Integer Literals

An integer literal can be assigned to an integer variable as long as it can fit into the variable. A compilation error would occur if the literal were too large for the variable to hold. For example, the statement byte b = 1000 would cause a compilation error, because 1000 cannot be stored in a variable of the byte type.

An integer literal is assumed to be of the int type, whose value is between -2^{31} (-2147483648) to $2^{31}-1$ (2147483647). To denote an integer literal of the long type, append it with the letter L or l. L is preferred because l (lowercase L) can easily be confused with 1 (the digit one).

Floating-Point Literals

Floating-point literals are written with a decimal point. By default, a floating-point literal is treated as a double type value. For example, 5.0 is considered a double value, not a float value. You can make a number a float by appending the letter f or F, and make a number a double by appending the letter d or D. For example, you can use 100.2f or 100.2F for a float number, and 100.2d or 100.2D for a double number.

Scientific Notation

Floating-point literals can also be specified in scientific notation, for example, 1.23456e+2, same as 1.23456e2, is equivalent to 123.456, and 1.23456e-2 is equivalent to 0.0123456. E (or e) represents an exponent and it can be either in lowercase or uppercase.

Shortcut Assignment Operators

Operator	*Example*	*Equivalent*
`+=`	`i += 8`	`i = i + 8`
`-=`	`f -= 8.0`	`f = f - 8.0`
`*=`	`i *= 8`	`i = i * 8`
`/=`	`i /= 8`	`i = i / 8`
`%=`	`i %= 8`	`i = i % 8`

INCREMENT AND DECREMENT OPERATORS

Operator	Name	Description
++var	preincrement evaluates	The expression (++var) increments var by 1 and to the *new* value in var *after* the increment.
var++	postincrement	The expression (var++) evaluates to the *original* value in var and increments var by 1.
--var	predecrement evaluates	The expression (--var) decrements var by 1 and to the *new* value in var *after* the decrement.
var--	postdecrement	The expression (var--) evaluates to the *original* value in var and decrements var by 1.

Increment and Decrement Operators, cont.

```
int i = 10;
int newNum = 10 * i++;
```

Same effect as

```
int newNum = 10 * i;
i = i + 1;
```

```
int i = 10;
int newNum = 10 * (++i);
```

Same effect as

```
i = i + 1;
int newNum = 10 * i;
```

Increment and Decrement Operators, cont.

Using increment and decrement operators makes expressions short, but it also makes them complex and difficult to read. Avoid using these operators in expressions that modify multiple variables, or the same variable for multiple times such as this: int k = ++i + i.

Arithmetic Operator Precedence

1	**()**	Parentheses
2	`++`	pre- or postfix increment
	`--`	pre- or postfix decrement
	`+ -`	unary plus, minus
3	`* / %`	multiplication, division, remainder
4	`+ -`	addition, substraction
5	=	assignment
	*= /= += -= %=	combinated assignment

Arithmetic Expressions

$$\boxed{\frac{3+4x}{5}-\frac{10(y-5)(a+b+c)}{x}+9(\frac{4}{x}+\frac{9+x}{y})}$$

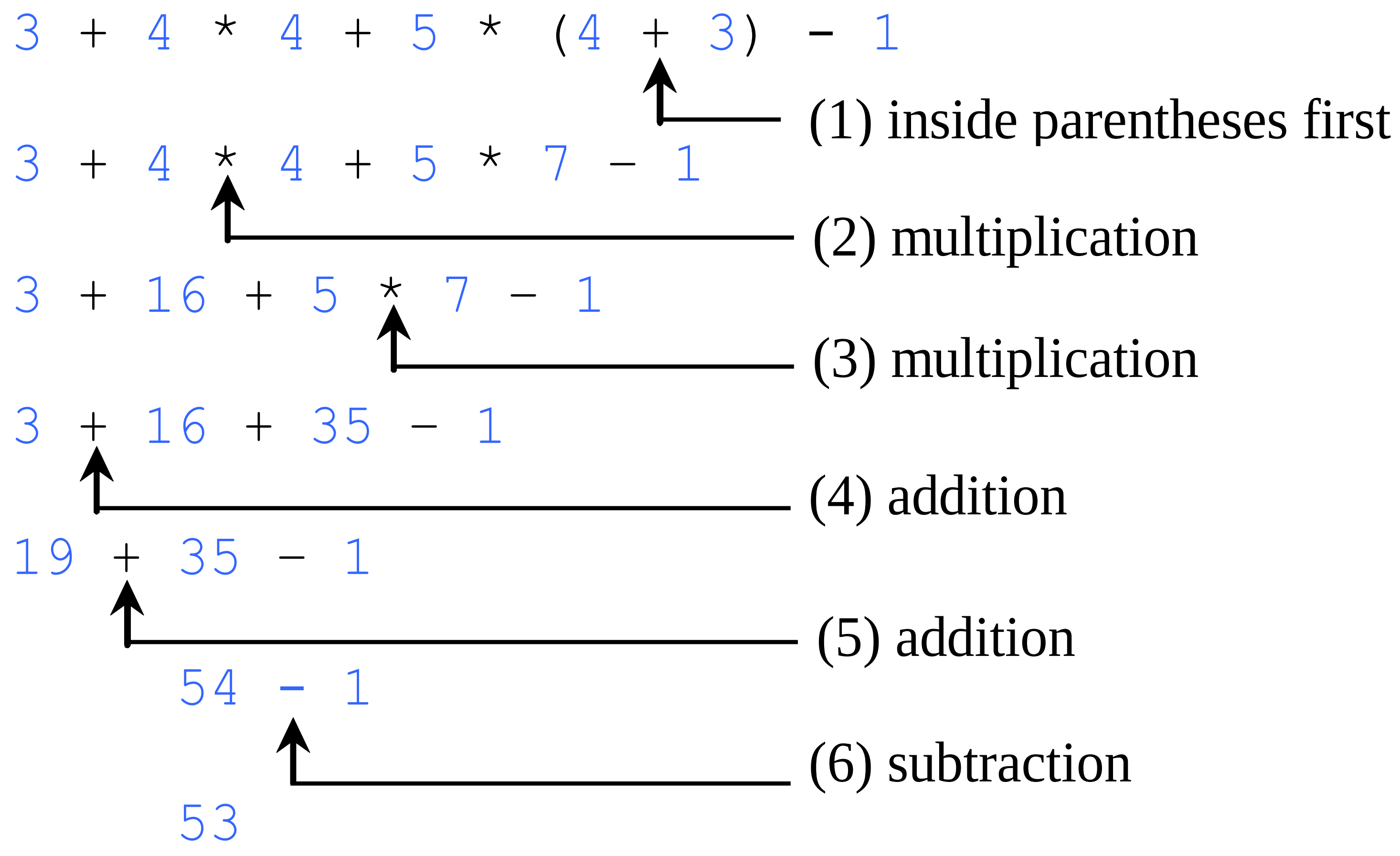
is translated to

(3+4*x)/5 – 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)

How to Evaluate an Expression

Though Java has its own way to evaluate an expression behind the scene, the result of a Java expression and its corresponding arithmetic expression are the same. Therefore, you can safely apply the arithmetic rule for evaluating a Java expression.

Evaluate an Expression



Problem: Converting Temperatures

Write a program that converts a Fahrenheit degree to Celsius using the formula:

$$celsius = (\tfrac{5}{9})(fahrenheit - 32)$$

Temp
fahrenheit : double
calcCels( ):void

Numeric Type Conversion

Consider the following statements:

```
byte i = 100;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```

Conversion Rules

When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:

1. If one of the operands is double, the other is converted into double.
2. Otherwise, if one of the operands is float, the other is converted into float.
3. Otherwise, if one of the operands is long, the other is converted into long.
4. Otherwise, both operands are converted into int.

Type Casting

Implicit casting

```
  double d = 3; (type widening)
```

Explicit casting

```
  int i = (int)3.0; (type narrowing)
  int i = (int)3.9; (Fraction part is
 truncated)
```

What is wrong? int x = 5 / 2.0;
int x = 2 / 4;

range increases

$\longrightarrow$

byte, short, int, long, float, double

Type Casting

- You can enlarge the size but you can not narrow it.
- If you have to narrow the size you have to use explicit casting.

Casting in an Augmented Expression

In Java, an augmented expression of the form **x1 op= x2** is implemented as **x1 = (T)(x1 op x2)**, where **T** is the type for **x1**. Therefore, the following code is correct.

int sum = **0**;

sum += **4.5**; // sum becomes 4 after this statement

sum += 4.5 is equivalent to **sum = (int)(sum + 4.5)**.

Problem: Computing Loan Payments

Write a java program that enter the interest rate, number of years, and loan amount, and computes monthly payment and total payment.

LoanPayments
interestRate : double yearNo : int loanAmount : double
compMonthlyPay( ) : void compTotalPay( ) : void

$$monthlyPayment = \frac{loanAmount \times monthlyInterestRate}{1 - \frac{1}{(1 + monthlyInterestRate)^{numberOfYears \times 12}}}$$

Character Data Type

Four hexadecimal digits.

```
char letter = 'A'; (ASCII)
char numChar = '4'; (ASCII)
char letter = '\u0041'; (Unicode)
char numChar = '\u0034'; (Unicode)
```

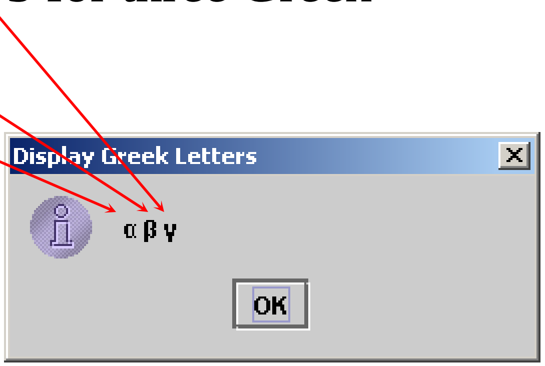
NOTE: The increment and decrement operators can also be used on char variables to get the next or preceding Unicode character. For example, the following statements display character b.

```
char ch = 'a';
System.out.println(++ch);
```

UNICODE FORMAT

Java characters use *Unicode*, a 16-bit encoding scheme established by the Unicode Consortium to support the interchange, processing, and display of written texts in the world’s diverse languages. Unicode takes two bytes, preceded by \u, expressed in four hexadecimal numbers that run from '\u0000' to '\uFFFF'. So, Unicode can represent `65535 + 1 characters.`

Unicode \u03b1 \u03b2 \u03b3 for three Greek letters



Escape Sequences for Special Characters

Description	*Escape Sequence*	*Unicode*
Backspace	`\b`	`\u0008`
Tab	`\t`	`\u0009`
Linefeed	`\n`	`\u000A`
Carriage return	`\r`	`\u000D`
Backslash	`\\`	`\u005C`
Single Quote	`\'`	`\u0027`
Double Quote	`\"`	`\u0022`

Casting between char and Numeric Types

```
int i = 'a'; // Same as int i = (int)'a';

char c = 97; // Same as char c = (char)97;
```

The String Type

The char type only represents one character. To represent a string of characters, use the data type called String. For example,

<u>String message = "Welcome to Java";</u>

<u>String</u> is actually a predefined class in the Java library just like the <u>System</u> class and <u>JOptionPane</u> class. The <u>String</u> type is not a primitive type. It is known as a *reference type*. Any Java class can be used as a reference type for a variable. For the time being, you just need to know how to declare a <u>String</u> variable, how to assign a string to the variable, and how to concatenate strings.

String Concatenation

// Three strings are concatenated
String message = "Welcome " + "to " + "Java";

// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2

// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B'; // s1 becomes SupplementB

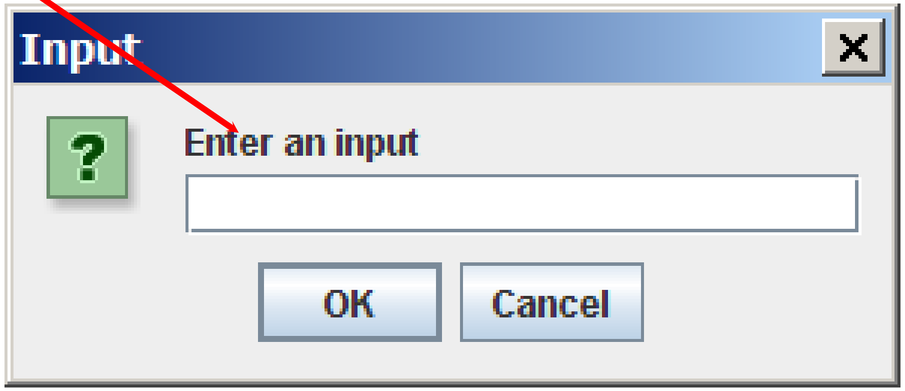
JOptionPane Input

This book provides two ways of obtaining input.

1. Using the Scanner class (console input)
2. Using JOptionPane input dialogs

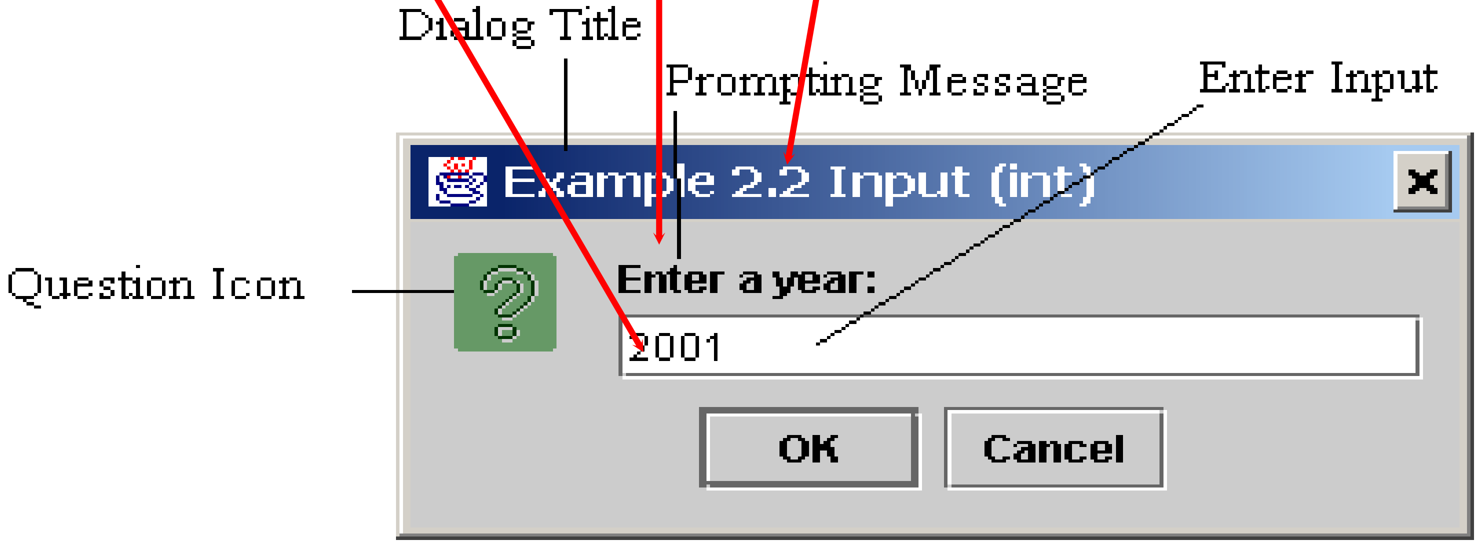
GETTING INPUT FROM INPUT DIALOG BOXES

```
String input =  JOptionPane.showInputDialog(
  "Enter an input");
```



Getting Input from Input Dialog Boxes

String string = JOptionPane.showInputDialog(
null, “Prompting Message”, “Dialog Title”,
JOptionPane.QUESTION_MESSAGE);



Two Ways to Invoke the Method

There are several ways to use the showInputDialog method. For the time being, you only need to know two ways to invoke it.

One is to use a statement as shown in the example:

```
String string = JOptionPane.showInputDialog(null, x,
  y, JOptionPane.QUESTION_MESSAGE);
```

where x is a string for the prompting message, and y is a string for the title of the input dialog box.

The other is to use a statement like this:

```
JOptionPane.showInputDialog(x);
```

where x is a string for the prompting message.

Converting Strings to Integers

The input returned from the input dialog box is a string. If you enter a numeric value such as 123, it returns “123”. To obtain the input as a number, you have to convert a string into a number.

To convert a string into an <u>int</u> value, you can use the static <u>parseInt</u> method in the <u>Integer</u> class as follows:

```
int intValue = Integer.parseInt(intString);
```

where <u>intString</u> is a numeric string such as “123”.

Converting Strings to Doubles

To convert a string into a double value, you can use the static parseDouble method in the Double class as follows:

```
double doubleValue
=Double.parseDouble(doubleString);
```

where doubleString is a numeric string such as "123.45".

Problem: Computing Loan Payments Using Input Dialogs

Same as the preceding program (slide 39) for computing loan payments, except that the input is entered from the input dialogs and the output is displayed in an output dialog.

$$\frac{loanAmount \times monthlyInterestRate}{1 - \frac{1}{(1 + monthlyInterestRate)^{numberOfYears \times 12}}}$$