Chapter 2 Elementary Programming

Objectives

- To write Java programs to perform simple calculations (§ 2.2).
- To use identifiers to name variables, constants, methods, and classes (§ 2.3).
- To use variables to store data (§ 2.4-2.5).
- To program with assignment statements and assignment expressions (§ 2.5).
- To use constants to store permanent data (§ 2.6).
- To declare Java primitive data types: byte, short, int, long, float, double, and char (§ 2.7-2.9).
- To use Java operators to write numeric expressions (§ 2.7-2.8).
- To represent characters using the char type (§ 2.9).
- To represent a string using the String type (§ 2.10).
- To obtain input from the console using the Scanner class (§ 2.11-2.12).
- To become familiar with Java documentation, programming style, and naming conventions (§ 2.13).
- To distinguish syntax errors, runtime errors, and logic errors (§ 2.14).
- To debug logic errors (§ 2.15).
- (GUI) To obtain input using the JOptionPane input dialog boxes (§ 2.16).

Introducing Programming with an Example

Computing the Area of a Circle

1. Read in the radius of a circle
2. Compute the area of the circle
3. Print out a message with the computed result
Variables

- A variable is a name for a location in memory used to hold a data value.
- Type, name and contents

Using a variable

- Declaring a variable – type and name
  - Instructs the compiler to reserve a portion of main memory to hold a particular type of value referred by a particular name
- Assign a value to a variable

Syntax: Variable Definition

typeName variableName;

Example:
int luckyNumber;

Purpose:
To define a new variable of a particular type

Syntax: Assignment

Syntax:
variableName = expression;

Example:
luckyNumber = 12;
luckyNumber = 5+7;

Purpose:
To assign a new value to a previously defined variable.
Data Types

- Fundamental or primitive data types and object types
- 8 primitive types
  - 6 number types: four integer types and two floating point types
  - 1 character type
  - 1 boolean type

Numerical Data Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Storage Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>(-2^7 \ (-128)) to (2^7\ -1 (127))</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>short</td>
<td>(-2^{15} \ (-32768)) to (2^{15}-1 (32767))</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>int</td>
<td>(-2^{31} \ (-2147483648)) to (2^{31}-1 (2147483647))</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>long</td>
<td>(-2^{63} \ (-9223372036854775808)) to (2^{63}-1 \ 9223372036854775807)</td>
<td>64-bit signed</td>
</tr>
<tr>
<td>float</td>
<td>Negative range: (-3.4028235E+38) to (-1.4E-45)</td>
<td>32-bit IEEE 754</td>
</tr>
<tr>
<td></td>
<td>Positive range: (1.4E-45) to (3.4028235E+38)</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>Negative range: (-1.7014118349223157E+308) to (-4.9E-324)</td>
<td>64-bit IEEE 754</td>
</tr>
<tr>
<td></td>
<td>Positive range: (4.9E-324) to (1.7014118349223157E+308)</td>
<td></td>
</tr>
</tbody>
</table>

Floating point numbers

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 0.0999999999999998, not 0.1. Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.
Number demo

An excellent tool to demonstrate how numbers are stored in a computer was developed by Richard Rasala. You can access it at

http://www.ccs.neu.edu/jpt/jpt_2_3/bitdisplay/applet.htm

Identifiers

- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).

Rules for Java identifier

- 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. (See Appendix A, “Java Keywords,” for a list of reserved words).
- An identifier cannot be true, false, or null.
- An identifier can be of any length.

Identifiers

- Conventions
  - variable names start with a lowercase letter
  - class names start with an uppercase letter
  - Meaningful names
  - Camel case
    - E.g. luckyNumber
Self Check

1. Which of the following are legal identifiers?
   - greeting1
   - g
   - void
   - 101dalmatians
   - Hello, World
   - greeting

2. Define a variable to hold your name. Use camel case in the variable name.

Declaring Variables - Examples

```c
int x;  // Declare x to be an integer variable;
double radius;  // Declare radius to be a double variable;
```

Syntax: Assignment

```c
variableName = expression;
```

Example:
```c
luckyNumber = 12;
luckyNumber = 5+7;
```

Purpose:
To assign a new value to a previously defined variable.
Assignment Statements

x = 1;  // Assign 1 to x;
radius = 1.0;  // Assign 1.0 to radius;


Declaring and Initializing in One Step

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


Expressions

- An expression is a combination of one or more operators and operands that perform a calculation
  - Operands might be numbers, variables, or other source of data
- Arithmetic expressions
### Numeric Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>34 + 1</td>
<td>35</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>34.0 – 0.1</td>
<td>33.9</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>300 * 30</td>
<td>9000</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>1.0 / 2.0</td>
<td>0.5</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td>20 % 3</td>
<td>2</td>
</tr>
</tbody>
</table>

### Division and remainder

- **Division operator**
  - Performs integer division when both operands are integers
  - 5 / 2 yields an integer 2.
  - 5.0 / 2 yields a double value 2.5
- **Remainder operator**
  - 5 % 2 yields 1 (the remainder of the division)

### Remainder Operator

How to determine whether a number is even or odd?

Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days?

- Saturday is the 6th day in a week.
- A week has 7 days.
- The 2nd day in a week is Tuesday.
- After 10 days, the day of the week is Tuesday.
Arithmetic Expressions

\[
\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)
\]

is translated to

\[
\frac{3+4x}{5} - 10\frac{(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)
\]

How to Evaluate an Expression

Arithmetic rules apply for evaluating a Java expression.

\[
3 + 4 \times 4 + 5 \times (4 + 3) - 1
\]

\[
3 + 4 \times 4 + 5 \times 7 - 1
\]

\[
3 + 16 + 5 \times 7 - 1
\]

\[
3 + 16 + 35 - 1
\]

\[
19 + 35 - 1
\]

\[
54 - 1
\]

\[
53
\]

(1) inside parentheses first
(2) multiplication
(3) multiplication
(4) addition
(5) addition
(6) subtraction

Compute Area Problem

```java
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);
    }
}
```
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);
    }
}

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);
    }
}

---

Literals

- A literal is a constant value that appears directly in the program.
- Number literal
  - double radius = 5.0;
  - int i = 34;
  - long x = 100000;
- String literal
  - System.out.println("Hello World");
Integer Literals

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

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.
  - E.g. 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.
  - E.g. you can use 100.2f or 100.2F for a float number, and 100.2d or 100.2D for a double number.
- Floating-point literals can also be specified in scientific notation
  - E (or e) represents an exponent and it can be either in lowercase or uppercase.
  - E.g. 1.23456e+2, same as 1.23456e2, is equivalent to 123.456
  - E.g. 1.23456e-2 is equivalent to 0.0123456.

The String Type

- String is a predefined class in the Java library.
  - The String type is a reference type. Any Java class can be used as a reference type for a variable (Chapter 7, "Objects and Classes.")
- String literal
  - System.out.print("Welcome to Java!");
  - Declare a String variable and assign a string to the variable
    - String message = "Welcome to Java";
    - System.out.println(message);
- Concatenate strings
String Concatenation

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

// String is concatenated with the values
System.out.println("The area for the circle of radius " + radius + " is " + area);
```

Constants

- A constant represents permanent data that never changes
- A constant must be declared and initialized in the same statement
- By convention, constants are named in uppercase

**Syntax:**
```
final datatype CONSTANTNAME = VALUE;
```

**Example:**
```
final double PI = 3.14159;
final int SIZE = 3;
```

Example – using constants

```java
public class ComputeArea {
    // Main method
    public static void main(String[] args) {
        final double PI = 3.14159; // declare a constant
        double radius;
        double area;
        // Assign a radius
        radius = 20;
        // Compute area
        area = radius * radius * PI;
        // Display results
        System.out.println("The area for the circle of radius " + radius + " is " + area);
    }
}
```
Problem: Displaying Time

Write a program that obtains hours and minutes from seconds.

DisplayTime.java

Problem: Converting Temperatures

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

\[ celsius = \left(\frac{5}{9}\right)(fahrenheit - 32) \]

FahrenheitToCelcius.java

Shortcut Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>i += 8</td>
<td>i = i + 8</td>
</tr>
<tr>
<td>-=</td>
<td>f -= 8.0</td>
<td>f = f - 8.0</td>
</tr>
<tr>
<td>*=</td>
<td>i *= 8</td>
<td>i = i * 8</td>
</tr>
<tr>
<td>/=</td>
<td>i /= 8</td>
<td>i = i / 8</td>
</tr>
<tr>
<td>%=</td>
<td>i %= 8</td>
<td>i = i % 8</td>
</tr>
</tbody>
</table>
## Increment and Decrement Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++var</td>
<td>preincrement</td>
<td>The expression (++var) increments var by 1 and evaluates to the new value in var after the increment.</td>
</tr>
<tr>
<td>var++</td>
<td>postincrement</td>
<td>The expression (var++) evaluates to the original value in var and increments var by 1.</td>
</tr>
<tr>
<td>--var</td>
<td>predecrement</td>
<td>The expression (--var) decrements var by 1 and evaluates to the new value in var after the decrement.</td>
</tr>
<tr>
<td>var--</td>
<td>postdecrement</td>
<td>The expression (var--) evaluates to the original value in var and decrements var by 1.</td>
</tr>
</tbody>
</table>

### Example

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

Same effect as

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

### Review

- Identifiers
- Variables
- Assignment statements
- Constants
- Numeric data types and operations
- The String type
- Obtaining input using Scanner class
Agenda

- Numeric type conversions
- Character data type and operations
- Other methods for obtaining user input
- Programming style and documentation
- Programming errors
- Debugging

Numeric Type Conversion

- It is helpful or necessary to convert a data value of one type to another type or mix data types in an expression

```java
double fahrenheit = 100;
double celsius = (5.0 / 9) * (fahrenheit - 32);
```

- Widening conversions: convert from one data type to another type that uses an equal or greater space
  - E.g. int to double
- Narrowing conversions: convert from one type to another type that uses less space
  - E.g. double to int

  range increases
  byte, short, int, long, float, double

Conversion

- Implicit
  - Promotion
  - Assignment conversion
- Explicit
  - Casting
Promotion

- Occurs automatically when a binary operation involving two operands of different types.
  
  ```java
double celsius = (5.0 / 9) * (fahrenheit - 32);
system.out.println("The celsius is "+ celsius);
```

- Conversion rules for numeric types:
  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.

Assignment Conversion

- Occurs when a value of one type is assigned to a variable of another type.
- Only widening conversion is allowed.
  
  ```java
double fahrenheit = 100; // widening, okay
int radius = 5.0; // narrowing, compiler error
int celsius = (5.0 / 9) * (fahrenheit - 32);
```

Explicit conversion

- Use the cast operator to convert a value to a specified type.
- Necessary for narrowing conversion.
- Cast operator has a higher precedence than multiplication & division.

```
(type) expression

Example:
int radius = (int) 5.0;
double smallRadius = (double) radius / 2;
```

Purpose:
To convert an expression to a different type.
Problem: Keeping Two Digits After Decimal Points

Write a program that displays the sales tax with two digits after the decimal point.

1. Initialize purchase amount and tax rate
2. Compute tax
3. Display tax with two digits after the decimal point

SalesTax.java

Character data type

- A fundamental data type used to store a single character
  - char letter = 'A';
- Encoding: convert a character to its binary representation
- Character set: a set of characters defined by an encoding scheme
  - Popular character sets
    - ASCII (8 bits)
      - 128 characters
      - Letters, punctuation, digits, common symbols, and accented characters
    - Unicode (16 bits)
      - 65536 characters
      - ASCII is a subset of Unicode set
      - Includes characters and symbols from many different languages
  - Java uses Unicode

Appendix B: ASCII Character Set

ASCII Character Set is a subset of the Unicode from \u0000 to \u007f

| \u0000 | \u0001 | \u0002 | \u0003 | \u0004 | \u0005 | \u0006 | \u0007 | \u0008 | \u0009 | \u000a | \u000b | \u000c | \u000d | \u000e | \u000f | \u0010 | \u0011 | \u0012 | \u0013 | \u0014 | \u0015 | \u0016 | \u0017 | \u0018 | \u0019 | \u001a | \u001b | \u001c | \u001d | \u001e | \u001f | \u0020 | \u0021 | \u0022 | \u0023 | \u0024 | \u0025 | \u0026 | \u0027 | \u0028 | \u0029 | \u002a | \u002b | \u002c | \u002d | \u002e | \u002f | \u0030 | \u0031 | \u0032 | \u0033 | \u0034 | \u0035 | \u0036 | \u0037 | \u0038 | \u0039 | \u003a | \u003b | \u003c | \u003d | \u003e | \u003f | \u0040 | \u0041 | \u0042 | \u0043 | \u0044 | \u0045 | \u0046 | \u0047 | \u0048 | \u0049 | \u004a | \u004b | \u004c | \u004d | \u004e | \u004f | \u0050 | \u0051 | \u0052 | \u0053 | \u0054 | \u0055 | \u0056 | \u0057 | \u0058 | \u0059 | \u005a | \u005b | \u005c | \u005d | \u005e | \u005f | \u0060 | \u0061 | \u0062 | \u0063 | \u0064 | \u0065 | \u0066 | \u0067 | \u0068 | \u0069 | \u006a | \u006b | \u006c | \u006d | \u006e | \u006f | \u0070 | \u0071 | \u0072 | \u0073 | \u0074 | \u0075 | \u0076 | \u0077 | \u0078 | \u0079 | \u007a | \u007b | \u007c | \u007d | \u007e | \u007f |
ASCII Character Set, cont.

ASCII Character Set is a subset of the Unicode from `\u0000` to `\u007f`.

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td><code>\u0021</code></td>
</tr>
<tr>
<td>#</td>
<td><code>\u0023</code></td>
</tr>
<tr>
<td>$</td>
<td><code>\u0024</code></td>
</tr>
<tr>
<td>%</td>
<td><code>\u0025</code></td>
</tr>
<tr>
<td>&amp;</td>
<td><code>\u0026</code></td>
</tr>
<tr>
<td>(</td>
<td><code>\u0028</code></td>
</tr>
<tr>
<td>)</td>
<td><code>\u0029</code></td>
</tr>
<tr>
<td>*</td>
<td><code>\u002A</code></td>
</tr>
<tr>
<td>+</td>
<td><code>\u002B</code></td>
</tr>
<tr>
<td>,</td>
<td><code>\u002C</code></td>
</tr>
<tr>
<td>.</td>
<td><code>\u002E</code></td>
</tr>
<tr>
<td>/</td>
<td><code>\u002F</code></td>
</tr>
<tr>
<td>0</td>
<td><code>\u0030</code></td>
</tr>
<tr>
<td>1</td>
<td><code>\u0031</code></td>
</tr>
<tr>
<td>2</td>
<td><code>\u0032</code></td>
</tr>
<tr>
<td>3</td>
<td><code>\u0033</code></td>
</tr>
<tr>
<td>4</td>
<td><code>\u0034</code></td>
</tr>
<tr>
<td>5</td>
<td><code>\u0035</code></td>
</tr>
<tr>
<td>6</td>
<td><code>\u0036</code></td>
</tr>
<tr>
<td>7</td>
<td><code>\u0037</code></td>
</tr>
<tr>
<td>8</td>
<td><code>\u0038</code></td>
</tr>
<tr>
<td>9</td>
<td><code>\u0039</code></td>
</tr>
<tr>
<td>:</td>
<td><code>\u003A</code></td>
</tr>
<tr>
<td>;</td>
<td><code>\u003B</code></td>
</tr>
<tr>
<td>&lt;</td>
<td><code>\u003C</code></td>
</tr>
<tr>
<td>=</td>
<td><code>\u003D</code></td>
</tr>
<tr>
<td>&gt;</td>
<td><code>\u003E</code></td>
</tr>
<tr>
<td>?</td>
<td><code>\u003F</code></td>
</tr>
</tbody>
</table>

ASCII Art

You can get creative using only 95 printable ASCII characters!

```
!"#$%&'()++,-./
0123456789:;<=?@
ABCDEFGHIJKLMNOPQRSTUVWXYZ`
abcdefghijklmnopqrstuvwxyz`
```

http://www.chris.com/ASCII/

Character data type

- A character literal is expressed with single quotes
  ```
  char letter = 'A';
  char numChar = '4';
  ```

- Character literal using Unicode
  ```
  char letter = '\u0041'; // character A
  char numChar = '\u0034'; // character 4
  ```

- String literal using Unicode
  ```
  String s = '\u0041\u0034'; // "A4"
  String msg = '\u6b22\u8fce \u03b1 \u03b2 \u03b3';
  ```
Example: Displaying Unicodes

"\u6B22\u8FCE \u03b1 \u03b2 \u03b3"

DisplayUnicode.java

---

Escape Sequences for Special Characters

<table>
<thead>
<tr>
<th>Description</th>
<th>Escape Sequence</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backspace</td>
<td>\b</td>
<td>\u0008</td>
</tr>
<tr>
<td>Tab</td>
<td>\t</td>
<td>\u0009</td>
</tr>
<tr>
<td>Linefeed</td>
<td>\n</td>
<td>\u000A</td>
</tr>
<tr>
<td>Carriage return</td>
<td>\r</td>
<td>\u000D</td>
</tr>
<tr>
<td>Backslash</td>
<td>\</td>
<td>\u005C</td>
</tr>
<tr>
<td>Single Quote</td>
<td>'</td>
<td>\u0027</td>
</tr>
<tr>
<td>Double Quote</td>
<td>&quot;</td>
<td>\u0022</td>
</tr>
</tbody>
</table>

---

Character Operations

- **Casting:** a char can be cast into any numeric type, and vice versa
  ```java
  char ch1 = 97;  // ch1 = 'a'
  char ch2 = (char) 97.25;  // ch2 = 'a'
  int i = 'a';  // i = 97
  ```
- **All numeric operators can be applied to char operands**
  ```java
  int i = '2' + '3';  // (int) '2' is 50; i=?
  ```
- **The + operator can be used to concatenate a char with a string**
  ```java
  System.out.println('2' + '3');  // displays '23'
  ```
- **The increment and decrement operators can be used to get the next or preceding Unicode character**
  ```java
  char ch = 'a';
  System.out.println(++ch);  // displays 'b'
  ```
Obtaining Input

1. Using Scanner class (§ 2.11)
2. Using JOptionPane input dialogs (§ 2.16)
3. Using command line arguments (§ 8.5)

Getting Input Using Scanner

1. Create a Scanner object
   
   ```java
   Scanner scanner = new Scanner(System.in);
   ```
   
   2. Use methods `next()`, `nextByte()`, `nextShort()`, `nextInt()`, `nextLong()`, `nextFloat()`, `nextDouble()`, or `nextBoolean()` to obtain a string, byte, short, int, long, float, double, or boolean value.

   ```java
   System.out.print("Enter a double value: ");
   Scanner scanner = new Scanner(System.in);
   double d = scanner.nextDouble();
   ```

   Example: TestScanner.java

Problem:
Computing Loan Payments

This program lets the user enter the interest rate, number of years, and loan amount and computes monthly payment and total payment.

\[
\text{loanAmount} \times \frac{1}{(1 + \text{monthlyInterestRate})^{\text{numberofYears} \times 12}}
\]

ComputeLoan.java
Getting Input from Command Line Arguments

- Command line arguments
  - `java ProgramName arg0 arg1 arg2 ...`
  - The command line arguments are passed to the main method through the parameter `args`
    ```java
    public static void main(String[] args)
    ```
  - We can use `args[0]`, `args[1]`, ... to access `arg0`, `arg1`, ... respectively
  - The arguments are stored as strings, we need to convert the arguments into numeric values when necessary

Converting Strings to Integers and Doubles

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

```java
int intValue = Integer.parseInt(intString);
```
where `intString` is a numeric string such as "123"

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

```java
double doubleValue = Double.parseDouble(doubleString);
```
where `doubleString` is a numeric string such as "123.45"

Example

Modify Compute Area program to reads radius from command line

```java
public class ComputeAreaCommand {
    public static void main(String[] args) {
        double radius = Double.parseDouble(args[0]);
        double area = radius * radius * 3.14;
        System.out.println("The area is "+ area);
    }
}
```

To run the program:

```
java ComputeAreaCommand 5.0
```
Getting Input from Input Dialog Boxes

- Using JOptionPane class to generate an input dialog and read in input
- Convert the input string to numeric types

```
String input = JOptionPane.showInputDialog(x);
```

where `x` is a string for the prompting message.

```
String input = 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.

Two Ways to Invoke Dialog Boxes

Problem: Computing Loan Payments Using Input Dialogs

Modify the previous program for computing loan payments, so that the input is entered from the input dialogs and the output is displayed in an output dialog.

```
ComputeLoanInputDialog.java
```
Problem: Monetary Units

This program lets the user enter the amount in decimal representing dollars and cents and output a report listing the monetary equivalent in single dollars, quarters, dimes, nickels, and pennies. Your program should report maximum number of dollars, then the maximum number of quarters, and so on, in this order.

```
ComputeChange.java
```

```
trace ComputeChange
Suppose amount is 11.56
remainingAmount = 1156
// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;
// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;
// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;
// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;
// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

```
animation ComputeChange
Suppose amount is 11.56
remainingAmount 1156
numberOfOneDollars 11
```

```
animation ComputeChange
Suppose amount is 11.56
remainingAmount 1156
numberOfOneDollars 11
```
```csharp
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

Suppose amount is 11.56
- numberOfOneDollars = 11
- numberOfQuarters = 2
- numberOfDimes = 0
- numberOfNickels = 0
- numberOfPennies = 6
Programming Style and Documentation

- Appropriate Comments
- Naming Conventions
- Proper Indentation and Spacing Lines
- Block Styles

Appropriate Comments

Include a summary at the beginning of the program to explain what the program does, its key features, and any unique techniques it uses.

Include comments for key steps explaining what they do.

Naming Conventions

- Choose meaningful and descriptive names.
- Variables and method names:
  - Use lowercase for the first word
  - Camel case: if the name consists of several words, concatenate all in one, capitalize the first letter of each subsequent word in the name.
    - E.g. radius, area, and purchaseAmount
- Class names:
  - Capitalize the first letter of each word in the name.
    - E.g. ComputeArea.
- Constants:
  - Capitalize all letters in constants, and use underscores to connect words.
    - E.g. PI and MAX_VALUE
Block Styles

Use end-of-line style for braces.

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Block Styles");
    }
}
```

Proper Indentation and Spacing

- **Indentation**
  - Indent two spaces.

- **Spacing**
  - Use blank line to separate segments of the code.

Programming Errors

- **Syntax Errors**
  - Detected by the compiler
- **Runtime Errors**
  - Causes the program to abort
- **Logic Errors**
  - Produces incorrect result
Syntax Errors

```java
public class ShowSyntaxErrors {
    public static void main(String[] args) {
        i = 30;
        System.out.println(i + 4);
    }
}
```

Runtime Errors

```java
public class ShowRuntimeErrors {
    public static void main(String[] args) {
        int i = 1 / 0;
    }
}
```

Logic Errors

```java
public class ShowLogicErrors {
    // Determine if a number is between 1 and 100 inclusively
    // Prompt the user to enter a number
    String input = JOptionPane.showInputDialog(null,
    "Please enter an integer:",
    "ShowLogicErrors", JOptionPane.QUESTION_MESSAGE);
    int number = Integer.parseInt(input);

    // Display the result
    System.out.println("The number is between 1 and 100, " +
    "inclusively? " + ((1 < number) && (number < 100)));
    System.exit(0);
}
```
Bug

A logic error is also called a bug

The first real bug found in a computer

Debugging

- Debugging is the process of finding and correcting errors
- Debugging approach
  - Hand-trace the program (i.e., catch errors by reading the program)
  - Insert print statements to show the values of the variables or the execution flow of the program
    ```java
double radius = Double.parseDouble(args[0]);
System.out.println("radius = " + radius);
double area = radius * radius * 3.14;
```
  - Use a debugger utility