Chapter 2 Elementary Programming
Introducing Programming with an Example

Listing 2.1 Computing the Area of a Circle

This program computes the area of the circle.

ComputeArea

Run
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);
    }
}
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);
    }
}

memory

radius 20
area 1256.636

print a message to the console
Reading Input from the Console

1. Create a Scanner object

   Scanner input = new Scanner(System.in);

2. Use the methods `next()`, `nextByte()`, `nextShort()`, `nextInt()`, `nextLong()`, `nextFloat()`, `nextDouble()`, or `nextBoolean()` to obtain to a string, `byte`, `short`, `int`, `long`, `float`, `double`, or `boolean` value. For example,

   System.out.print("Enter a double value: ");
   Scanner input = new Scanner(System.in);
   double d = input.nextDouble();
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.
Variables

// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + 
 area + " for radius "+radius);

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + 
 area + " for radius "+radius);
Declaring Variables

```c
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

```c
int x = 1;
double d = 1.4;
```
Constants

final datatype CONSTANTNAME = VALUE;
final double PI = 3.14159;
final int SIZE = 3;
## 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>$-2^{63}$ to $2^{63}$−1</td>
<td>64-bit signed</td>
</tr>
<tr>
<td></td>
<td>(i.e., −9223372036854775808 to 9223372036854775807)</td>
<td></td>
</tr>
</tbody>
</table>
| 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

<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>
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)
Remainder Operator

Remainder is very useful in programming. For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property 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? You can find that day is Tuesday using the following expression:

Saturday is the 6th day in a week
A week has 7 days
After 10 days
(6 + 10) % 7 is 2
The 2nd day in a week is Tuesday
Problem: Displaying Time

Write a program that obtains hours and minutes from seconds.
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.
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:

```plaintext
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, $1.23456\times10^2$, same as $1.23456\times10^2$, is equivalent to 123.456, and $1.23456\times10^{-2}$ is equivalent to 0.0123456. $E$ (or $e$) represents an exponent and it can be either in lowercase or uppercase.
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

\[
(3+4\times x)/5 - 10\times(y-5)\times(a+b+c)/x + 9\times(4/x + (9+x)/y)
\]
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.

$$3 \, + \, 4 \, \times \, 4 \, + \, 5 \, \times \, (4 \, + \, 3) \, - \, 1$$

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

3 \, + \, 4 \, \times \, 4 \, + \, 5 \, \times \, 7 \, - \, 1$

3 \, + \, 16 \, + \, 5 \, \times \, 7 \, - \, 1$

3 \, + \, 16 \, + \, 35 \, - \, 1$

19 \, + \, 35 \, - \, 1$

54 \, - \, 1$

53
Problem: Converting Temperatures

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

\[ celsius = \left(\frac{5}{9}\right)(fahrenheit - 32) \]
Problem: Displaying Current Time

Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.

The `currentTimeMillis` method in the `System` class returns the current time in milliseconds since the midnight, January 1, 1970 GMT. (1970 was the year when the Unix operating system was formally introduced.) You can use this method to obtain the current time, and then compute the current second, minute, and hour as follows.

```
ShowCurrentTime
```

```
Run
```

```
Unix Epoch
01-01-1970
00:00:00 GMT
```

```
ElapsedTime
```

```
Time
```

```
Current Time
System.currentTimeMillis()
```

```
### Shortcut Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+=</code></td>
<td><code>i += 8</code></td>
<td><code>i = i + 8</code></td>
</tr>
<tr>
<td><code>-=</code></td>
<td><code>f -= 8.0</code></td>
<td><code>f = f - 8.0</code></td>
</tr>
<tr>
<td><code>*=</code></td>
<td><code>i *= 8</code></td>
<td><code>i = i * 8</code></td>
</tr>
<tr>
<td><code>/=</code></td>
<td><code>i /= 8</code></td>
<td><code>i = i / 8</code></td>
</tr>
<tr>
<td><code>%=</code></td>
<td><code>i %= 8</code></td>
<td><code>i = i % 8</code></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 \texttt{var} by 1 and evaluates to the \textit{new} value in \texttt{var after} the increment.</td>
</tr>
<tr>
<td>var++</td>
<td>postincrement</td>
<td>The expression (var++) evaluates to the \textit{original} value in \texttt{var} and increments \texttt{var} by 1.</td>
</tr>
<tr>
<td>--var</td>
<td>predecrement</td>
<td>The expression (--var) decrements \texttt{var} by 1 and evaluates to the \textit{new} value in \texttt{var after} the decrement.</td>
</tr>
<tr>
<td>var--</td>
<td>postdecrement</td>
<td>The expression (var--) evaluates to the \textit{original} value in \texttt{var} and decrements \texttt{var} by 1.</td>
</tr>
</tbody>
</table>
Increment and Decrement Operators, cont.

```c
int i = 10;
int newNum = 10 * i++; // Same effect as
int newNum = 10 * i;
i = i + 1;
```

```c
int i = 10;
int newNum = 10 * (i++); // Same effect as
int newNum = 10 * i;
i = i + 1;
```

```c
int i = 10;
int newNum = 10 * ++i; // Same effect as
int newNum = 10 * i;
i = i + 1;
```
Numeric Type Conversion

Consider the following statements:

```java
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)
Character Data Type

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
Problem: Displaying Unicodes

Write a program that displays two Chinese characters and three Greek letters.

DisplayUnicode
Run
# 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>
Casting between char and Numeric Types

```java
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,

```java
String message = "Welcome to Java";
```

String is actually a predefined class in the Java library just like the System class and JOptionPane class. The String 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. Reference data types will be thoroughly discussed in Chapter 7, “Objects and Classes.” For the time being, you just need to know how to declare a String 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
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, its supporting data structures, and any unique techniques it uses.

Include your name, class section, instructor, date, and a brief description at the beginning of the program.
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.
Proper Indentation and Spacing

_indentation

– Indent two spaces.

_spacing

– Use blank line to separate segments of the code.
Use end-of-line style for braces.

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

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

- Syntax Errors
  - Detected by the compiler
- Runtime Errors
  - Causes the program to abort
- Logic Errors
  - Produces incorrect result
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;
    }
}
```
import java.util.Scanner; // Scanner is in java.util
public class ShowLogicErrors {
    // Determine if a number is between 1 and 100 inclusively
    public static void main(String[] args) {
        // Prompt the user to enter a number
        Scanner input = new Scanner(System.in);
        System.out.print("Enter an integer: ");
        int number = input.nextInt();

        // Display the result
        System.out.println("The number is between 1 and 100 inclusively? " + ((1 > number) && (number > 100)));
        System.exit(0);
    }
}
Logic errors are called *bugs*. The process of finding and correcting errors is called debugging. A common approach to debugging is to use a combination of methods to narrow down to the part of the program where the bug is located. You can hand-trace the program (i.e., catch errors by reading the program), or you can insert print statements in order to show the values of the variables or the execution flow of the program. This approach might work for a short, simple program. But for a large, complex program, the most effective approach for debugging is to use a debugger utility.
Debugger

Debugger is a program that facilitates debugging. You can use a debugger to

- Execute a single statement at a time.
- Trace into or stepping over a method.
- Set breakpoints.
- Display variables.
- Display call stack.
- Modify variables.
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");
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:

```java
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:

```java
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 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”.

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:

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

where doubleString is a numeric string such as “123.45”.

Problem: Computing Loan Payments Using Input Dialogs

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

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

Run