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).

Numerical Data Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Storage Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>$-2^7$ to $2^7 - 1$ (-128 to 127)</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>short</td>
<td>$-2^{15}$ to $2^{15} - 1$ (-32768 to 32767)</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>int</td>
<td>$-2^{31}$ to $2^{31} - 1$ (-2147483648 to 2147483647)</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>long</td>
<td>$-2^{63}$ to $2^{63} - 1$ (i.e., -9223372036854775808 to 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: -1.7976931348623157E+308 to -4.9E-324</td>
<td>64-bit IEEE 754</td>
</tr>
<tr>
<td></td>
<td>Positive range: 4.9E-324 to 1.7976931348623157E+308</td>
<td></td>
</tr>
</tbody>
</table>

```
int i = 5;
double d = 5.5;
```
Number Literals

A *literal* is a constant value that appears directly in the program. For example: 34; 1000000; and 5.0 are literals in the following statements:

```java
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.
Character Data Type

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

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
Appendix B: ASCII Character Set

ASCII- American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>Table B.1 ASCII Character Set in the Decimal Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
<tr>
<td>nul soh stx ext esc esc qck bel bs hts</td>
</tr>
<tr>
<td>nl return formFeed cr so si dle dc1 dc2 dc3</td>
</tr>
<tr>
<td>ds tab sp alr quot und asel fs gs</td>
</tr>
<tr>
<td>lft ctrl alt prn brk scr harls esc elem</td>
</tr>
<tr>
<td>del</td>
</tr>
</tbody>
</table>

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. 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 8, “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.
Variables

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

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

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
(Initializing)

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;
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. (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.

Named Constants

```java
final datatype CONSTANTNAME = Value;
final double PI = 3.14159;
final int SIZE = 3;
```
Naming Conventions

Choose meaningful and descriptive names.

Variables 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 variable `programmingCourse`

Constants:
Capitalize all letters in constants, and use underscores to connect words. Example, the constant `PI` and `MAX_VALUE`

Escape Sequences for Special Characters

<table>
<thead>
<tr>
<th>Description</th>
<th>Escape Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backspace</td>
<td>\b</td>
</tr>
<tr>
<td>Tab</td>
<td>\t</td>
</tr>
<tr>
<td>Linefeed</td>
<td>\n</td>
</tr>
<tr>
<td>Carriage return</td>
<td>\r</td>
</tr>
<tr>
<td>Backslash</td>
<td>\</td>
</tr>
<tr>
<td>Single Quote</td>
<td>'</td>
</tr>
<tr>
<td>Double Quote</td>
<td>&quot;</td>
</tr>
</tbody>
</table>
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

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:

\[(6 + 10) \% 7 = 2\]

Introducing Programming with an Example

Listing 2.1 Computing the Area of a Circle

This program computes the area of the circle.
Trace a Program Execution

```java
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);
    }
}
```

Liang, Introduction to Java Programming, Ninth Edition, (c) 2013 Pearson Education, Inc. All rights reserved.
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

Trace a Program Execution

Liang, Introduction to Java Programming, Ninth Edition, (c) 2013 Pearson Education, Inc. All rights reserved.
Trace a Program Execution

```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);
    }
}
```

Reading Input from the Console

1. Create a Scanner object
   ```java
   Scanner input = new Scanner(System.in);
   ```

2. Use the methods `next()`, `nextByte()`, `nextShort()`, `nextInt()`, `nextLong()`, `nextFloat()`, `nextDouble()`, or `nextBoolean()` to obtain a string, `byte`, `short`, `int`, `long`, `float`, `double`, or `boolean` value. For example,
   ```java
   System.out.print("Enter a double value: ");
   Scanner input = new Scanner(System.in);
   double d = input.nextDouble();
   ```
Problem: Displaying Time

Write a program that obtains hours and minutes from seconds.
Exponent Operations

```java
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
```

Arithmetic Expressions

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

is translated to

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

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

1. Inside parentheses first
2. Multiplication
3. Multiplication
4. Addition
5. Addition
6. Subtraction

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>

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

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

```
int i = 10;
int newNum = 10 * (++i);
i = i + 1;
int newNum = 10 * i;
```  
Same effect as
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:
```java
int k = ++i + i;
```

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

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)

What is wrong? int x = 5 / 2.0;

range increases

byte, short, int, long, float, double
Casting in an Augmented Expression

In Java, an augmented expression of the form $x_1 \ op= x_2$ is implemented as $x_1 = (T)(x_1 \ op \ x_2)$, where $T$ is the type for $x_1$. Therefore, the following code is correct.

```java
int sum = 0;
sum += 4.5; // sum becomes 4 after this statement
```

$\text{sum} += 4.5$ is equivalent to $\text{sum} = (\text{int})(\text{sum} + 4.5)$.

String Concatenation

```java
// 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
```
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;
```

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”.

---

Debugging

- 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.
- 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.