Chapter 2
Elementary Programming

2.1 Introduction 34
2.2 Writing a Simple Program 34
2.3 Reading Input from the Console 37
2.4 Identifiers 40
2.5 Variables 40
2.6 Assignment Statements and Assignment Expressions 42
2.7 Named Constants 43
2.8 Naming Conventions 44
2.9 Numeric Data Types and Operations 45
2.10 Numeric Literals 48
2.11 JShell 50
2.12 Evaluating Expressions and Operator Precedence 52
2.13 Case Study: Displaying the Current Time 54
2.14 Augmented Assignment Operators 56
2.15 Increment and Decrement Operators 57
2.16 Numeric Type Conversions 58
2.17 Software Development Process 61
2.18 Case Study: Counting Monetary Units 64
2.19 Common Errors and Pitfalls 67
Chapter 2
Elementary Programming

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).
- To read a `byte`, `short`, `int`, `long`, `float`, or `double` value from the keyboard (§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 use JShell to quickly test Java code (§2.11).
- To write and evaluate numeric expressions (§2.12).
- To obtain the current system time using `System.currentTimeMillis()` (§2.13).
- To use augmented assignment operators (§2.14).
- To distinguish between `postincrement` and `preincrement` and between `postdecrement` and `predecrement` (§2.15).
- To cast the value of one type to another type (§2.16).
- To describe the software development process and apply it to develop the loan payment program (§2.17).
- To write a program that converts a large amount of money into smaller units (§2.18).
- To avoid common errors and pitfalls in elementary programming (§2.19).
Chapter 2
Elementary Programming

2.1 Introduction 34

- You will learn elementary programming using Java primitive data types and related subjects, such as variables, constants, operators, expressions, and input and output.

2.2 Writing a Simple Program 34

- Writing a program involves designing algorithms and data structures, as well as translating algorithms into programming code.
- An Algorithm describes how a problem is solved in terms of the actions to be executed, and it specifies the order in which the actions should be executed.
- Algorithms can be described in natural languages or in pseudocode (natural language mixed with some programming code).
- Computing an area of a circle. The algorithm for this program can be described as follows:
  1. Read in the Radius
  2. Compute the area using the following formula:
     \[ \text{Area} = \text{radius} \times \text{radius} \times \pi \]
  3. Display the area.

- Java provides data types for representing integers, floating-point numbers, characters, and Boolean types. These types are known as primitive data types.
- When you code, you translate an algorithm into a programming language understood by the computer. The outline of the program is:

```java
public class ComputeArea {
    public static void main(String[] args) {
        double radius; // Declare radius
        double area; // Declare area

        // Assign a radius
        radius = 20; // New value is radius

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

        // Display results
        System.out.println("The area for the circle of radius "+radius + " is " + area);
    }
}
```
The program needs to **declare** a symbol called a variable that will represent the radius. **Variables** are used to store data and computational results in the program.

Use descriptive names rather than x and y. Use radius for radius, and area for area. Specify their data types to let the compiler know what radius and area are, indicating whether they are integer, float, or something else.

The program declares radius and area as double-precision variables. The reserved word **double** indicates that radius and area are double-precision floating-point values stored in the computer.

For the time being, we will assign a fixed number to radius in the program. Then, we will compute the area by assigning the expression `radius * radius * 3.14159` to area.

The program’s output is:

The area for the circle of radius 20.0 is 1256.636

A string constant should not cross lines in the source code. Use the **concatenation** operator (+) to overcome such problem.
2.3 Reading Input from the Console 37

Getting Input Using Scanner

- Create a Scanner object

```java
Scanner scanner = new Scanner(System.in);
```

- 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 scanner = new Scanner(System.in);
double d = scanner.nextDouble();
```

LISTING 2.2 ComputeAreaWithConsoleInput.java

```java
import java.util.Scanner; // Scanner is in the java.util package

public class ComputeAreaWithConsoleInput {
    public static void main(String[] args) {
        // Create a Scanner object
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter a radius
        System.out.print("Enter a number for radius: ");
        double radius = input.nextDouble();

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

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

- Caution
  By default a Scanner object reads a string separated by whitespaces (i.e. ‘ ‘, ‘\t’, ‘\f’, ‘\r’, and ‘\n’).
2.4 Identifiers 40

- Programming languages use special symbols called *identifiers* to name such programming entities as variables, constants, methods, classes, and packages.

- The following are the rules for naming 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.

- For example:
  - Legal identifiers are for example: $2, ComputeArea, area, radius, and showMessageDialog.
  - Illegal identifiers are for example: 2A, d+4.
  - Since Java is **case-sensitive**, X and x are different identifiers.
2.5 Variables 40

- Variables are used to store data in a program.
- You can write the code shown below to compute the area for different radius:

```java
// 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

- Variables are used for representing data of a certain type.
- To use a variable, you declare it by telling the compiler the name of the variable as well as what type of data it represents. This is called variable declaration.
- Declaring a variable tells the compiler to allocate appropriate memory space for the variable based on its data type. The following are examples of variable declarations:

```java
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;
```

- If variables are of the same type, they can be declared together using short-hand form:

```java
Datatype var1, var2, ..., varn;   // variables are separated by commas
```

Declaring and Initializing Variables in One Step

- You can declare a variable and initialize it in one step.

```java
int x = 1;
```

This is equivalent to the next two statements:

```java
int x;
x = 1;
```

// shorthand form to declare and initialize vars of same type
int i = 1, j = 2;

- **Tip:** A variable must be declared before it can be assigned a value.
2.6 Assignment Statements and Assignment Expressions 42

- After a variable is declared, you can assign a value to it by using an assignment statement. The syntax for assignment statement is:

  \[
  \text{variable} = \text{expression};
  \]

  \[
  x = 1; \quad // \text{Assign 1 to } x; \quad \Rightarrow \text{Thus } 1 = x \text{ is wrong}
  \]

  \[
  \text{radius} = 1.0; \quad // \text{Assign 1.0 to radius;}
  \]

  \[
  a = 'A'; \quad // \text{Assign 'A' to } a;
  \]

  \[
  x = 5 \times (3 / 2) + 3 \times 2; \quad // \text{Assign the value of the expression to } x;
  \]

  \[
  x = y + 1; \quad // \text{Assign the addition of } y \text{ and 1 to } x;
  \]

- The variable can also be used in the expression.

  \[
  x = x + 1; \quad // \text{the result of } x + 1 \text{ is assigned to } x;
  \]

- To assign a value to a variable, the variable name must be on the left of the assignment operator.

  \[
  l = x; \quad // \text{would be wrong}
  \]

- In Java, an assignment statement can also be treated as an expression that evaluates to the value being assigned to the variable on the left-hand side of the assignment operator. For this reason, an assignment statement is also known as an assignment expression, and the symbol = is referred to as the assignment operator.

  \[
  \text{System.out.println(}x = 1)\text{;}
  \]

  which is equivalent to

  \[
  x = 1;
  \text{System.out.println(}x\text{;}
  \]

  If a value is assigned to multiple variables, you can use chained assignments like this:

  \[
  i = j = k = 1;
  \]

  which is equivalent to

  \[
  k = 1; \quad j = k; \quad i = j;
  \]
2.7 Named Constants 43

- The value of a variable may change during the execution of the program, but a constant represents permanent data that never change.
- The syntax for declaring a constant:

  ```java
  final datatype CONSTANTNAME = value;
  ```

  ```java
  final double PI = 3.14159; // Declare a constant
  final int SIZE = 3;
  ```

- A constant must be declared and initialized before it can be used. You cannot change a constant’s value once it is declared. By convention, constants are named in uppercase.

LISTING 2.4 ComputeAreaWithConstant.java

```java
import java.util.Scanner; // Scanner is in the java.util package

public class ComputeAreaWithConstant {
    public static void main(String[] args) {
        final double PI = 3.14159; // Declare a constant
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter a radius
        System.out.print("Enter a number for radius: ");
        double radius = input.nextDouble();

        // Compute area
        double area = radius * radius * PI;

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

- **Note:** There are three benefits of using constants:
  - You don’t have to repeatedly type the same value.
  - The value can be changed in a single location.
  - The program is easy to read.
2.8 Naming Conventions

- Use **lowercase** for variables and methods. If a 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. Ex: `showInputDialog`.
- Choose **meaningful** and descriptive names. For example, the variables radius and area, and the method `computeArea`.
- **Capitalize** the first letter of each word in the **class** name. For example, the class name `ComputeArea`.
- Capitalize all letters in **constants**. For example, the constant `PI` and `MAX_VALUE`.
- Do **not** use class names that are already used in Java library. For example, since the `System` class is defined in Java, you should not name your class `System`. 
2.9 Numeric Data Types and Operations 45

2.9.1 Numeric Types

- Every data type has a range of **values**. The compiler allocates memory space to store each variable or constant according to its data type.
- Java has six numeric types: four for integers and two for floating-point numbers.

**TABLE 2.1 Numeric 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 \text{ to } 2^7 - 1) ((-128 \text{ to } 127))</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>short</td>
<td>(-2^{15} \text{ to } 2^{15} - 1) ((-32768 \text{ to } 32767))</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>int</td>
<td>(-2^{31} \text{ to } 2^{31} - 1) ((-2147483648 \text{ to } 2147483647))</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>long</td>
<td>(-2^{63} \text{ to } 2^{63} - 1) (i.e., (-9223372036854775808 \text{ to } 9223372036854775807))</td>
<td>64-bit signed</td>
</tr>
<tr>
<td>float</td>
<td>Negative range: (-3.4028235E+38 \text{ to } -1.4E-45) Positive range: (1.4E-45 \text{ to } 3.4028235E+38)</td>
<td>32-bit IEEE 754</td>
</tr>
<tr>
<td>double</td>
<td>Negative range: (-1.7976931348623157E+308 \text{ to } -4.9E-324) Positive range: (4.9E-324 \text{ to } 1.7976931348623157E+308)</td>
<td>64-bit IEEE 754</td>
</tr>
</tbody>
</table>

2.9.2 Reading Numbers from the Keyboard

```java
Scanner input = new Scanner(System.in);
int value = input.nextInt();
```

**TABLE 2.2 Methods for Scanner Objects**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextByte()</td>
<td>reads an integer of the <strong>byte</strong> type.</td>
</tr>
<tr>
<td>nextShort()</td>
<td>reads an integer of the <strong>short</strong> type.</td>
</tr>
<tr>
<td>nextInt()</td>
<td>reads an integer of the <strong>int</strong> type.</td>
</tr>
<tr>
<td>nextLong()</td>
<td>reads an integer of the <strong>long</strong> type.</td>
</tr>
<tr>
<td>nextFloat()</td>
<td>reads a number of the <strong>float</strong> type.</td>
</tr>
<tr>
<td>nextDouble()</td>
<td>reads a number of the <strong>double</strong> type.</td>
</tr>
</tbody>
</table>
2.9.3 Numerical Operators

TABLE 2.3 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>

- \( \frac{5}{2} \) yields an integer (2)
- \( \frac{5.0}{2} \) yields a double value (2.5)
- \( -\frac{5}{2} \) yields an integer value (-2)
- \( -\frac{5.0}{2} \) yields a double value (-2.5)
- \( 5 \% 2 \) yields 1 (the remainder of the division.)
- \( -7 \% 3 \) yields -1
- \( -12 \% 4 \) yields 0
- \( -26 \% -8 \) yields -2
- \( 20 \% -13 \) yields 7

- **Remainder** is very useful in programming. For example, an even number \( \% 2 \) is always 0 and an odd number \( \% 2 \) is always 1. Thus, 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 it in 10 days? You can find that day is Tuesday using the following expression:

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

Saturday is the 6\text{th} day in a week

A week has 7 days

The 2\text{nd} day in a week is Tuesday

After 10 days
• The program in Listing 2.5 (DisplayTime.java) obtains minutes and remaining seconds from an amount of time in seconds. For example, 500 seconds contains 8 minutes and 20 seconds.

LISTING 2.5 DisplayTime.java

```java
import java.util.Scanner;

public class DisplayTime {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        // Prompt the user for input
        System.out.print("Enter an integer for seconds: ");
        int seconds = input.nextInt();
        int minutes = seconds / 60; // Find minutes in seconds
        int remainingSeconds = seconds % 60; // Seconds remaining
        System.out.println(seconds + " seconds is " + minutes + " minutes and " + remainingSeconds + " seconds");
    }
}
```

Enter an integer for seconds: 500
500 seconds is 8 minutes and 20 seconds

2.9.4 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
```
2.10 Numeric Literals

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

  ```java
type i = 34;
long l = 1000000;
double d = 5.0;
```

2.10.1 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. (Note the range for a byte value is from –128 to 127.)
- 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 (lowercase L).
- For example, the following code display the decimal value 65535 for hexadecimal number FFFF.

  ```java
  System.out.println(0x FFFF);
  ```

2.10.2 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.
- **The double type values are more accurate** than float type values.

  ```java
  System.out.println("1.0 / 3.0 is "+ 1.0 / 3.0);
  // displays 1.0 / 3.0 is 0.3333333333333333
  System.out.println("1.0F / 3.0F is "+ 1.0F / 3.0F);
  // displays 1.0F / 3.0F is 0.33333334
  ```

2.10.3 Scientific Notations

- 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.
2.11 JShell 50

- JShell is a command line interactive tool introduced in Java 9.
- JShell enables you to type a single Java statement and get it executed to see the result right away without having to write a complete class.
- This feature is commonly known as REPL (Read-Evaluate-Print Loop), which evaluates expressions and executes statements as they are entered and shows the result immediately.

  - Open a Command Window and type jshell to launch JShell

  ![Figure 2.2 JShell is launched.](image)

  - You can enter a Java statement from the jshell prompt. For example, enter int x = 5

  ![Figure 2.3 Enter a Java statement at the jshell command prompt](image)

  - To print the variable, simply type x. Alternatively, you can type System.out.println(x)

  ![Figure 2.4 Print a variable](image)
2.12 Evaluating Expressions and Operator Precedence 52

- For example, the arithmetic expression

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

can be translated into a Java expression as:

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

- Operators contained within pairs of **parentheses** are evaluated first.
- Parentheses can be **nested**, in which case the expression in the **inner** parentheses is evaluated first.
- Multiplication, division, and remainder operators are applied next. Order of operation is applied from left to right. Addition and subtraction are applied last.

**LISTING 2.6 FahrenheitToCelsius.java**

```java
import java.util.Scanner;

public class FahrenheitToCelsius {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter a degree in Fahrenheit: ");
        double fahrenheit = input.nextDouble();

        // Convert Fahrenheit to Celsius
        double celsius = (5.0 / 9) * (fahrenheit - 32);
        System.out.println("Fahrenheit " + fahrenheit + " is " + celsius + " in Celsius");
    }
}
```

```
Enter a degree in Fahrenheit: 100
Fahrenheit 100.0 is 37.77777777777778 in Celsius
```
2.13 Case Study: Displaying the Current Time 54

- Write a program that displays current time in GMT (Greenwich Mean Time) 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.

![Diagram](image)

**FIGURE 2.8** The `System.currentTimeMillis()` return the number of milliseconds since the Unix epoch.

**LISTING 2.7** ShowCurrentTime.java

```java
public class ShowCurrentTime {
    public static void main(String[] args) {
        // Obtain the total milliseconds since midnight, Jan 1, 1970
        long totalMilliseconds = System.currentTimeMillis();

        // Obtain the total seconds since midnight, Jan 1, 1970
        long totalSeconds = totalMilliseconds / 1000;

        // Compute the current second in the minute in the hour
        long currentSecond = totalSeconds % 60;

        // Obtain the total minutes
        long totalMinutes = totalSeconds / 60;

        // Compute the current minute in the hour
        long currentMinute = totalMinutes % 60;

        // Obtain the total hours
        long totalHours = totalMinutes / 60;

        // Compute the current hour
        long currentHour = totalHours % 24;

        // Display results
        System.out.println("Current time is "+ currentHour + ":" + currentMinute + ":" + currentSecond + " GMT");
    }
}
Current time is 17:31:26 GMT
```
### 2.14 Augmented Assignment Operators

#### Table 2.4 Augmented Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Example</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>Addition assignment</td>
<td><code>i += 8</code></td>
<td><code>i = i + 8</code></td>
</tr>
<tr>
<td>-=</td>
<td>Subtraction assignment</td>
<td><code>i -= 8</code></td>
<td><code>i = i - 8</code></td>
</tr>
<tr>
<td>*=</td>
<td>Multiplication assignment</td>
<td><code>i *= 8</code></td>
<td><code>i = i * 8</code></td>
</tr>
<tr>
<td>/=</td>
<td>Division assignment</td>
<td><code>i /= 8</code></td>
<td><code>i = i / 8</code></td>
</tr>
<tr>
<td>%=</td>
<td>Remainder assignment</td>
<td><code>i %= 8</code></td>
<td><code>i = i % 8</code></td>
</tr>
</tbody>
</table>
2.15 Increment and Decrement Operators 57

- There are two more shortcut operators for incrementing and decrementing a variable by 1. These two operators are ++, and --. They can be used in prefix or suffix notations.

```
suffix  x++;  // Same as x = x + 1;
prefix  ++x;  // Same as x = x + 1;
suffix  x--;  // Same as x = x - 1;
prefix  --x;  // Same as x = x - 1;
```

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
<th>Example (assume i = 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>++var</td>
<td>preincrement</td>
<td>Increment var by 1, and use the new var value in the statement</td>
<td>int j = ++i;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>// j is 2, i is 2</td>
</tr>
<tr>
<td>var++</td>
<td>postincrement</td>
<td>Increment var by 1, but use the original var value in the statement</td>
<td>int j = i++;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>// j is 1, i is 2</td>
</tr>
<tr>
<td>--var</td>
<td>predecrement</td>
<td>Decrement var by 1, and use the new var value in the statement</td>
<td>int j = --i;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>// j is 0, i is 0</td>
</tr>
<tr>
<td>var--</td>
<td>postdecrement</td>
<td>Decrement var by 1, and use the original var value in the statement</td>
<td>int j = i--;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>// j is 1, i is 0</td>
</tr>
</tbody>
</table>

Ex:\n\n\n\n```
int i = 10;  
int newNum = 10 * i++;  // Same effect as  
\nint newNum = 10 * i;  
i = i + 1;
```

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

- Using increment and decrement operators make expressions short; 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.
2.16 Numeric Type Conversions 58

- Consider the following statements:

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

Are these statements correct?

- 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.

- Thus, the result of 1 / 2 is 0, and the result of 1.0 / 2 is 0.5.

- **Type Casting** is an operation that converts a value of one data type into a value of another data type.

- Casting a variable of a type with a small range to variable with a larger range is known as **widening** a type. Widening a type can be performed **automatically** without explicit casting.

- Casting a variable of a type with a large range to variable with a smaller range is known as **narrowing** a type. Narrowing a type must be performed **explicitly**.

- **Caution:** Casting is necessary if you are assigning a value to a variable of a smaller type range. A compilation error will occur if casting is not used in situations of this kind. Be careful when using casting. Lost information might lead to inaccurate results.

```java
float f = (float) 10.1;
int i = (int) f;

double d = 4.5;
int i = (int) d;  // d is not changed
System.out.println("d " + d + " i " + i); // answer is d 4.5 i 4
```

**Implicit casting**

double d = 3;  // type widening

**Explicit casting**

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

What is wrong?

```java
int i = 1;
byte b = i;  // Error because explicit casting is required
```
2.17 Software Development Process 61

- The software development life cycle is a multistage process that includes requirements specification, analysis, design, implementation, testing, deployment, and maintenance.

![Software Development Life Cycle Diagram]

**FIGURE 2.9** At any stage of the software development life cycle, it may be necessary to go back to a previous stage to correct errors or deal with other issues that might prevent the software from functioning as expected.

- **Requirement Specification**: A formal process that seeks to understand the problem and document in detail what the software system needs to do. This phase involves close interaction between users and designers. Most of the examples in this book are simple, and their requirements are clearly stated. In the real world, however, problems are not well defined. You need to study a problem carefully to identify its requirements.

- **System Analysis**: Seeks to analyze the business process in terms of data flow, and to identify the system’s input and output. Part of the analysis entails modeling the system’s behavior. The model is intended to capture the essential elements of the system and to define services to the system.

- **System Design**: The process of designing the system’s components. This phase involves the use of many levels of abstraction to decompose the problem into manageable components, identify classes and interfaces, and establish relationships among the classes and interfaces.

- **IPO**: The essence of system analysis and design is input, process, and output.

- **Implementation**: The process of translating the system design into programs. Separate programs are written for each component and put to work together. This phase requires the use of a programming language like Java. The implementation involves coding, testing, and debugging.

- **Testing**: Ensures that the code meets the requirements specification and weeds out bugs. An independent team of software engineers not involved in the design and implementation of the project usually conducts such testing.

- **Deployment**: Deployment makes the project available for use. For a Java program, this means installing it on a desktop or on the Web.
• **Maintenance**: Maintenance is concerned with changing and improving the product. A software product must continue to perform and improve in a changing environment. This requires periodic upgrades of the product to fix *newly* discovered bugs and incorporate changes.

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

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

**LISTING 2.9 ComputeLoan.java**

```java
import java.util.Scanner;

public class ComputeLoan {
    public static void main(String[] args) {
        // Create a Scanner
        Scanner input = new Scanner(System.in);

        // Enter yearly interest rate
        System.out.print("Enter yearly interest rate, for example 8.25: ");
        double annualInterestRate = input.nextDouble();

        // Obtain monthly interest rate
        double monthlyInterestRate = annualInterestRate / 1200;

        // Enter number of years
        System.out.print("Enter number of years as an integer, for example 5: ");
        int numberOfYears = input.nextInt();

        // Enter loan amount
        System.out.print("Enter loan amount, for example 120000.95: ");
        double loanAmount = input.nextDouble();

        // Calculate payment
        double monthlyPayment = loanAmount * monthlyInterestRate / (1 - 1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12));
        double totalPayment = monthlyPayment * numberOfYears * 12;

        // Display results
        System.out.println("The monthly payment is "+(int)(monthlyPayment * 100) / 100.0);
        System.out.println("The total payment is "+(int)(totalPayment * 100) / 100.0);
    }
}
```

Enter yearly interest rate, for example 8.25: 5.75
Enter number of years as an integer, for example 5: 15
Enter loan amount, for example 120000.95: 250000
The monthly payment is 2076.02
The total payment is 373684.53
2.18 Case Study: Counting 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.

LISTING 2.10 ComputeChange.java

```java
import java.util.Scanner;

public class ComputeChange {
    public static void main(String[] args) {
        // Create a Scanner
        Scanner input = new Scanner(System.in);

        // Receive the amount
        System.out.print("Enter an amount in double, for example 11.56: ");
        double amount = input.nextDouble();

        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;

        // Display results
        String output = "Your amount \n" + amount + " consists of \n" + "dollars\n" + "quarters\n" + "dimes\n" + "nickels\n" + "pennies";

        System.out.println(output);
    }
}
```

Enter an amount in double, for example 11.56: 11.56
Your amount 11.56 consists of
11 dollars
2 quarters
0 dimes
1 nickels
1 pennies
2.19 Common Errors and Pitfalls

- Common Error 1: Undeclared/Uninitialized Variables and Unused Variables
  - Java is case-sensitive, X and x are different identifiers
    ```java
double interestRate = 0.05;
double interest = interestRate * 45;
// error: cannot find symbol interestRate
```

- Common Error 2: Integer Overflow
  - Max 32 bit integer value is 2147483647
    ```java
    int value = 2147483647 + 1;
    // value will actually be -2147483648
    ```

- Common Error 3: Round-off Errors
  - Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy. For example,
    - Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.
      ```java
      System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
      // display 0.5000000000000000
      System.out.println(1.0 - 0.9);
      // display 0.09999999999999998, not 0.9
      ```

- Common Error 4: Unintended Integer Division

- Common Pitfall 1: Redundant Input Objects

  ```java
  int number1 = 1;
  int number2 = 2;
  double average = (number1 + number2) / 2;
  System.out.println(average);
  ```

  (a) Bad code: display 1

  ```java
  int number1 = 1;
  int number2 = 2;
  double average = (number1 + number2) / 2.0;
  System.out.println(average);
  ```

  (b) Good code: display 1.5

  ```java
  Scanner input1 = new Scanner(System.in);
  System.out.print("Enter an integer: ");
  int v1 = input1.nextInt();
  Scanner input2 = new Scanner(System.in);
  System.out.print("Enter a double value: ");
  double v2 = input2.nextDouble();
  ```

  (a) Bad code: two input objects

  ```java
  Scanner input = new Scanner(System.in);
  System.out.print("Enter an integer: ");
  int v1 = input.nextInt();
  System.out.print("Enter a double value: ");
  double v2 = input.nextDouble();
  ```

  (b) Good code: one input object