Chapter 2
Elementary Programming

2.1 Introduction

- 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 Simple Programs

- 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.
- 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
// ComputeArea.Java: compute the area of a circle Comment
public class ComputeArea   // Class Name
{
   public static void main(String[] args)// Main Method signature
   {
      double radius; // Data type & variable
      double area;

      // Assign a radius
      radius = 20;

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

      // 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

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);
    }
}
```

Enter a number for radius: 23
The area for the circle of radius 23.0 is 1661.90111

- Caution

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

- 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

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

```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
```
2.6 Assignment Statements and Assignments Expressions

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

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

  \[
  x = 1; \quad \text{// Assign 1 to x; } \Rightarrow \text{Thus 1 = x is wrong}
  \]
  \[
  \text{radius} = 1.0; \quad \text{// Assign 1.0 to radius;}
  \]
  \[
  a = \text{'}A\text{'}; \quad \text{// Assign 'A' to a;}
  \]
  \[
  x = 5 * (3 / 2) + 3 * 2; \quad \text{// Assign the value of the expression to x;}
  \]
  \[
  x = y + 1; \quad \text{// Assign the addition of y and 1 to x;}
  \]

- The variable can also be used in the expression.

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

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

  \[
  l = x \text{ would be wrong.}
  \]

- 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.

  \[
  \text{System.out.println(x = 1);} \quad \text{which is equivalent to}
  \]
  \[
  x = 1; \quad \text{System.out.println(x);} \quad \text{The following statement is also correct:}
  \]
  \[
  i = j = k = 1; \quad \text{which is equivalent to}
  \]
  \[
  k = 1; j = k; i = j;
  \]
Declaring and Initializing Variables in One Step

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

  ```
  int x = 1;
  ```

  This is equivalent to the next two statements:

  ```
  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.
2.7 Named Constants

- 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:

```
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.

// ComputeArea.java: compute the area of a circle Comment
```
public class ComputeArea       // Class Name
{
    public static void main(String[] args) // Main Method signature
    {
        final double PI = 3.14159;  // declare a constant
        double radius = 20;       // assign a radius

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

        // Display results
        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 Numerical Data Types and Operations

- 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>
<thead>
<tr>
<th>Name</th>
<th>Storage Size</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>$2^7 \text{--} -2^7$ (-128) to $2^7 - 1$ (127)</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>$2^{15} \text{--} -2^{15}$ (-32768) to $2^{15} - 1$ (32767)</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>$2^{32} \text{--} -2^{32}$ (-2147483648) to $2^{32} - 1$ (2147483647)</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>$2^{63} \text{--} -2^{63}$ to $2^{63} - 1$</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>6 - 7 significant digits of accuracy</td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>14 - 15 significant digits of accuracy</td>
</tr>
</tbody>
</table>

2.8.1 Numerical Operators

+, -, *, /, and %

- $5/2$ yields an integer \(2\)
- $5.0/2$ yields a double value \(2.5\)
- $-5/2$ yields an integer value \(-2\)
- $-5.0/2$ 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. So you can use this property to determine whether a number is even or odd. Suppose you know January 1, 2005 is Saturday, you can find that the day for February 1, 2005 is Tuesday using the following expression:

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

\[
(6 + 31) \% 7 \text{ is } 2
\]

A week has 7 days

January has 31 days

The 2\text{nd} day in a week is Tuesday
• LISTING 2.4 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 + ", " + remainingSeconds + " seconds");
    }
}
```

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

• A unary operator has only one operand. A binary operator has two operands.

• NOTE

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

```java
System.out.println(1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
```

displays 0.5000000000000001, not 0.5, and

```java
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.
2.8.2 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
int i = 34;
long l = 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` (lowercase L).
- For example, the following code display the decimal value 65535 for hexadecimal number FFFF.

```java
System.out.println(0xFFFF);
```

**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
```

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

**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.
2.8.3 Evaluating Java Expressions

- 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:

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

- 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 2.5 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
2.9 Problem: Displaying the Current Time

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

- **Listing 2.6** `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 = (int)(totalSeconds % 60);

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

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

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

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

        // Display results
        System.out.println("Current time is " + currentHour + ":" + currentMinute + ":" + currentSecond + " GMT");
    }
}
```

Current time is 17:31:26 GMT
2.10 Shorthand Operators

Table 2.4 Shorthand 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>

- 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.

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

Ex:

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

Same effect as:

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

Ex:

double x = 1.0;
double y = 5.0;
double z = x-- + (++y);

After execution, y = 6.0, z = 7.0, and x = 0.0;

- 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`.
2.11 Numeric Type Conversions

- 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**
```
int i = (int)3.0; // type narrowing
```

What is wrong?

```java
int i = 1;
byte b = i;  // Error because explicit casting is required
```
2.12 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.

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

- LISTING 2.8 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
2.13 Character Data Type and Operations

- The character data type, char, is used to represent a single character.
- A character literal is enclosed in **single** quotation marks.

```java
char letter = 'A'; // Assigns A to char variable letter (ASCII)
char numChar = '4'; // Assigns numeric character 4 to numChar (ASCII)
```

**Caution:** A string literal must be enclosed in quotation marks. A character literal is a single character enclosed in single quotation marks. So “A” is a **string**, and ‘A’ is a **character**.

### 2.13.1 Unicode and ASCII code

- Java uses 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 (See the Unicode Web site at [www.unicode.org](http://www.unicode.org) for more information.)
- Unicode takes **two** bytes, preceded by \u, expressed in four hexadecimal digits that run from ‘\u0000’ to ‘\uFFFF’. For example, the “coffee” is translated into Chinese using two characters. The Unicode of these two characters are “\u5496\u5561”.

```java
char letter = '\u0041'; // Unicode ➔ 16-bit encoding scheme
char numChar = '\u0034'; (Unicode)
```

- Unicode can represent 65,536 characters, since FFFF in hexadecimal is 65535.
- Most computer ASCII (American Standard Code for Information Interchange), a **7-bit** encoding scheme for representing all uppercase and lowercase letter, digits, punctuation marks, and control characters.
- Unicode includes ASCII code with ‘\u0000’ to ‘\u007F’ corresponding to **128** ASCII characters. (See Appendix B).
- **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);
```
### 2.13.2 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>

- Suppose you want to print the **quoted** message show below:

  He said “Java is fun”

  Here is how to write the statement:

  ```java
  System.out.println("He said \"Java is fun\"\");
  ```
2.13.3 Casting between char and Numeric Types

- A char can be cast into any numeric type, and vice versa.
- Implicit casting can be used if the result of a casting fits into the target variable. Otherwise explicit casting must be used.
- All numeric operation can be applied to the char operands.
- The char operand is cast into a number if the other operand is a number or a character.
- If the other operand is a string, the character is concatenated with the string.

```java
int i = 'a'; // Same as int i = (int)'a'; // (int) a is 97
char c = 99; // Same as char c = (char)99;

int i = '1' + '2'; // (int) 1 is 49 and (int) 2 is 50
System.out.println("i is " + i);

int j = 1 + 'a'; // (int) a is 97
System.out.println("j is " + 98);
System.out.println(j + " is the Unicode for character " + (char) j);
System.out.println("Chapter " + 2);
```

Output is:

```
i is 99
j is 98
98 is the Unicode for character b
Chapter 2
```
2.14 Problem: 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.

- LIST 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 " + amount + " consists of 
        " + numberOfOneDollars + " dollars" + 
        " + numberOfQuarters + " quarters" + 
        " + numberOfDimes + " dimes" + 
        " + numberOfNickels + " nickels" + 
        " + numberOfPennies + " 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
2.15 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 6, “Classes and Objects.” 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

- The plus sign (+) is the concatenation operator if one of the operands is a string.

- If one of the operands is a non-string (e.g. a number), the non-string value is converted into a string and concatenated with the other string.

  ```java
  // Three strings are concatenated
  String message = "Welcome " + "to " + "Java";
  message += " and Java is fun";  // message = Welcome to Java and Java is fun
  
  // 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';    // s becomes SupplementB
  ```

```java
i = 1; j = 3;
System.out.println("i + j is " + i + j);  \(\Rightarrow i + j \) is 13
System.out.println("i + j is " + (i + j)); \(\Rightarrow i + j \) is 4
```
2.16 Programming Style and Documentation

- Programming Style deals with what programs look like.
- Documentation is the body of explanatory remarks and comments pertaining to a program.
- Programming style and documentation are as important as coding. They make the programs easy to read.

2.16.1 Appropriate Comments and Comments Style

- Include a summary at the beginning of the program to explain what the program does, its key features, its supporting data structures, and unique techniques it uses.
- In a long program, you should also include comments that introduce each major step and explain anything that is difficult to read.
- Make your comments concise to they do not crowd the program or make it difficult to read.
- Include your name, class section, date, instruction, and a brief description at the beginning of the program.

2.16.2 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.
- Do not use class names that are already used in Java library. For example, the constants PI and MAX_VALUE.

2.16.3 Proper Indentation and Spacing Lines

- Indentation is used to illustrate the structural relationships between program’s components or statements.
- Indent two spaces in each subcomponent more than the structure which it is nested.
- Use a single space on both sides of a binary operator.

```java
boolean b = 3 + 4 * 4 > 5 * (4 + 3)
```

- Use a blank line to separate segments of the code.
2.16.4 Block Styles

- A block is a group of statements surrounded by braces. Use end-of-line style for braces or next-line style.

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

End-of-line style

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

Next-line style
2.17 Programming Errors

2.17.1 Syntax Errors “Compilation Error”
- Errors that occur during compilation are called syntax errors or compilation errors.
- Syntax errors result from errors in code construction, such as mistyping a keyword, omitting some necessary punctuation, or using an opening brace without a corresponding closing brace.
- These errors are easily detected, because the compiler tells you where they are and the reasons for them.

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

2.17.2 Runtime Errors
- Runtime errors are errors that cause a program to terminate abnormally.
- Runtime errors occur while an application is running where the environment detects an operation that is impossible to carry out.
- For instance, an input error occurs when the user enters an unexpected input value that the program can’t handle. To prevent input errors, the program should prompt the user to enter the correct type of values.
- Another example of a run time error is division by zero.

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

2.17.3 Logic Errors
- Logic errors occur when a program doesn’t perform the way it was intended to.
- For example, the program doesn’t have syntax or runtime errors, but it does not print the correct result.

```
// ShowLogicErrors.java: The program contains a logic error
// Suppose you wrote the following program to add number1 to number2
import javax.swing.JOptionPane;
public class ShowLogicErrors {
    public static void main(String[] args) {
        // Add number1 to number2
        int number1 = 3;
        int number2 = 3;
        number2 += number1 + number2;
        System.out.println("number2 is " + number2);
    }
}
```
2.17.4 Debugging

- Finding logic errors “bugs” is challenging and the process of finding and correcting errors is called debugging.
- You can hand-trace 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.
2.18 (GUI) Getting Input from Input Dialog Boxes

- 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.

  Example:
  ```java
  String input =
      JOptionPane.showInputDialog(
          null, 
          "Enter a year:",
          "Example 2.2 Input (int)",
          JOptionPane.QUESTION_MESSAGE
      );
  ```

- The other is to use a statement like this:

  ```java
  JOptionPane.showMessageDialog(x);
  ```

  where x is a string for the prompting message.

  Example:
  ```java
  String input =
      JOptionPane.showInputDialog(
          "Enter an input"
      );
  ```
2.18.1 Converting String to Numbers

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:

```
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:

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

where `doubleValue` is a numeric string such as “123.45”.
2.18.2 Using Input Dialog Boxes

LISTING 2.11 ComputeLoanUsingInputDialog.java

```java
import javax.swing.JOptionPane;

public class ComputeLoanUsingInputDialog {
    public static void main(String[] args) {
        // Enter yearly interest rate
        String annualInterestRateString = JOptionPane.showInputDialog("Enter yearly interest rate, for example 8.25: ");
        double annualInterestRate = Double.parseDouble(annualInterestRateString);
        // Obtain monthly interest rate
        double monthlyInterestRate = annualInterestRate / 1200;
        // Enter number of years
        String numberOfYearsString = JOptionPane.showInputDialog("Enter number of years as an integer, for example 5: ");
        int numberOfYears = Integer.parseInt(numberOfYearsString);
        // Enter loan amount
        String loanString = JOptionPane.showInputDialog("Enter loan amount, for example 120000.95: ");
        double loanAmount = Double.parseDouble(loanString);
        // Calculate payment
        double monthlyPayment = loanAmount * monthlyInterestRate / (1 - 1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12));
        double totalPayment = monthlyPayment * numberOfYears * 12;
        // Format to keep two digits after the decimal point
        monthlyPayment = (int)(monthlyPayment * 100) / 100.0;
        totalPayment = (int)(totalPayment * 100) / 100.0;
        // Display results
        String output = "The monthly payment is " + monthlyPayment + " The total payment is " + totalPayment;
        JOptionPane.showMessageDialog(null, output);
    }
}
```