Chapter 2 Elementary Programming

2.1 Introduction

The focus of this chapter is on learning elementary programming techniques to solve problems.

2.2 Writing a Simple Program

Writing a program involves designing a strategy for solving the problem and then using a programming language to implement that strategy.

Writing a program involves designing algorithms and translating algorithms into programming instructions, or code.

An algorithm describes how a problem is solved by listing the actions that need to be taken and the order of their execution. Algorithms can help the programmer plan a program before writing it in a programming language.

Algorithms can be described in natural languages or in pseudocode (natural language mixed with some programming code).

It’s always good practice to outline your program (or its underlying problem) in the form of an algorithm before you begin coding.

Variable

Rather than using x and y as variable names, choose descriptive names: in this case, radius for radius, and area for area.

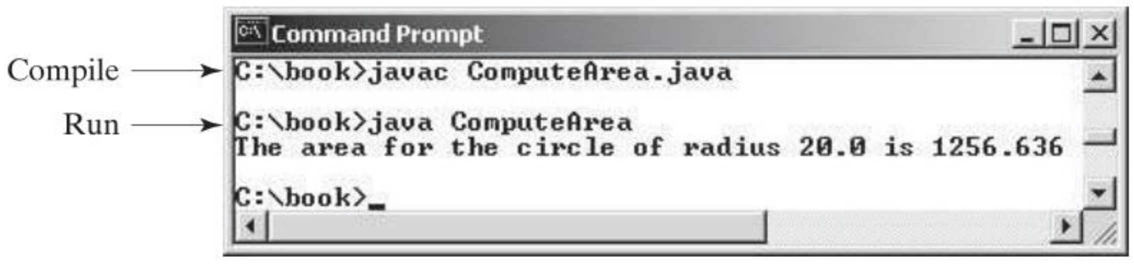
To let the compiler know what radius and area are, specify their data types. That is the kind of data stored in a variable, whether integer, real number, or something else. This is known as declaring variables.

Java provides simple data types for representing integers, real numbers, characters, and Boolean types. These types are known as primitive data types or fundamental types.

Real numbers (i.e., numbers with a decimal point) are represented using a method known as floating-point in computers. So, the real numbers are also called floating-point numbers. In Java, you can use the keyword double to declare a floating-point variable.

LISTING 2.1 ComputeArea.java

```
1   public class ComputeArea {
2     public static void main(String[] args) {
3       double radius; // Declare radius
4       double area; // Declare area
5
6       // Assign a radius
7       radius = 20; // radius is now 20
8
9       // Compute area
10      area = radius * radius * 3.14159;
11
12      // Display results
13      System.out.println("The area for the circle of radius " +
14        radius + " is " + area);
15    }
16  }
```



```
C:\book>javac ComputeArea.java

C:\book>java ComputeArea
The area for the circle of radius 20.0 is 1256.636

C:\book>_
```

FIGURE 2.1 The program displays the area of a circle.

declare variable

assign value

tracing program

concatenate strings

The plus sign (+) has two meanings: one for addition and the other for concatenating (combining) strings. The plus sign (+) in lines 13–14 is called a string concatenation operator. It combines two strings into one. If a string is combined with a number, the number is converted into a string and concatenated with the other string. Therefore, the plus signs (+) in lines 13–14 concatenate strings into a longer string, which is then displayed in the output.

2.3 Reading Input from the Console

Reading input from the console enables the program to accept input from the user.

Scanner class for console input.

System.out to refer to the standard output device

System.in to the standard input device.

default, the output device is the display monitor and the input device is the keyboard.

console output, use the println method

Console input is not directly supported in Java, but you can use the Scanner class to create an object to read input from System.in

```
Scanner input = new Scanner(System.in);
```

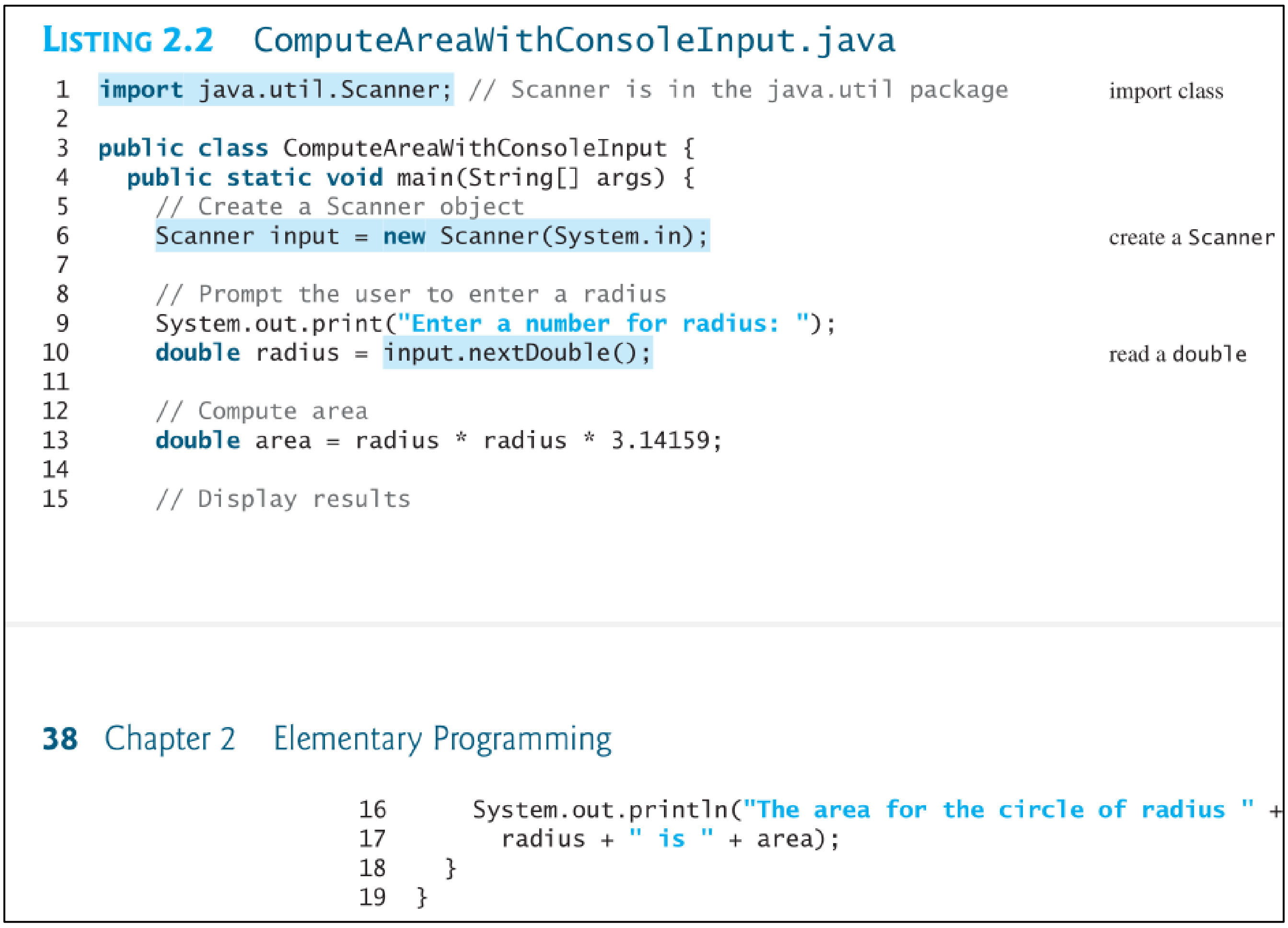
The syntax new Scanner(System.in) creates an object of the Scanner type.

The syntax Scanner input declares that input is a variable whose type is Scanner.

The whole line Scanner input = new Scanner(System.in) creates a Scanner object and assigns its reference to the variable input.

An object may invoke its methods. To invoke a method on an object is to ask the object to perform a task. You can invoke the nextDouble() method to read a double value as follows:

```
double radius = input.nextDouble();
```



LISTING 2.2 ComputeAreaWithConsoleInput.java

```
 1  import java.util.Scanner; // Scanner is in the java.util package        import class
 2
 3  public class ComputeAreaWithConsoleInput {
 4    public static void main(String[] args) {
 5      // Create a Scanner object
 6      Scanner input = new Scanner(System.in);                               create a Scanner
 7
 8      // Prompt the user to enter a radius
 9      System.out.print("Enter a number for radius: ");
10      double radius = input.nextDouble();                                   read a double
11
12      // Compute area
13      double area = radius * radius * 3.14159;
14
15      // Display results
```

```
16      System.out.println("The area for the circle of radius " +
17        radius + " is " + area);
18    }
19  }
```

Line 9: prompt, because it directs the user to enter an input. Your program should always tell the user what to enter when expecting input from the keyboard.

println moves to the beginning of the next line after displaying the string, but print does not advance to the next line when completed.

The Scanner class is in the java.util package.

Two types of import statements:

Specific / Explicit import

specifies a single class in the import statement.

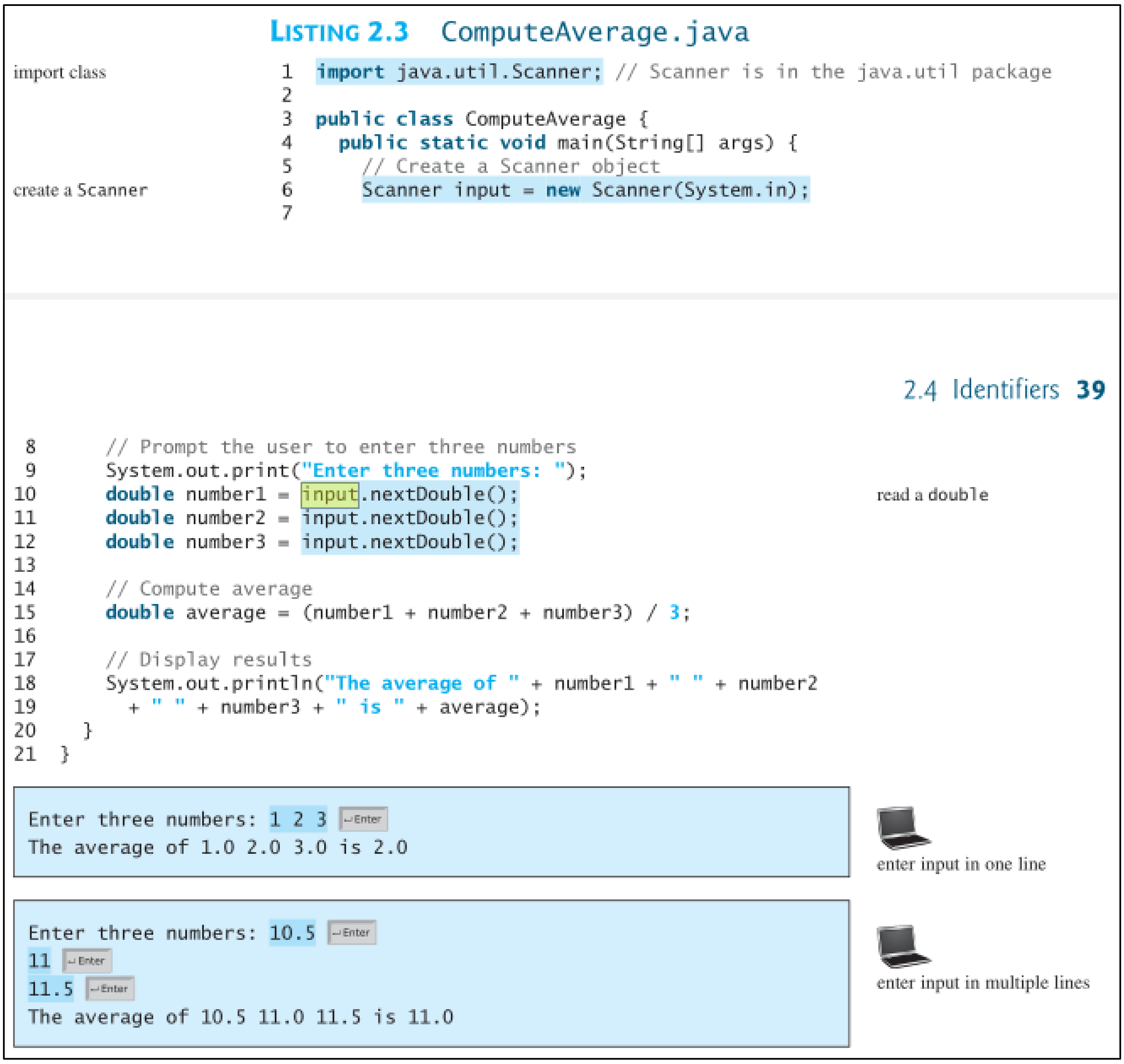
```
import java.util.Scanner;
```

wildcard / Implicit import

all the classes in a package by using the asterisk as the wildcard.

```
import java.uitl.*;
```

no performance difference between a specific import and a wildcard import declaration.



LISTING 2.3 ComputeAverage.java

import class

create a Scanner

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

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

```

read a double

```java
    // Prompt the user to enter three numbers
    System.out.print("Enter three numbers: ");
    double number1 = input.nextDouble();
    double number2 = input.nextDouble();
    double number3 = input.nextDouble();

    // Compute average
    double average = (number1 + number2 + number3) / 3;

    // Display results
    System.out.println("The average of " + number1 + " " + number2
      + " " + number3 + " is " + average);
  }
}
```

```
Enter three numbers: 1 2 3 ↵Enter
The average of 1.0 2.0 3.0 is 2.0
```

enter input in one line

```
Enter three numbers: 10.5 ↵Enter
11 ↵Enter
11.5 ↵Enter
The average of 10.5 11.0 11.5 is 11.0
```

enter input in multiple lines

You may enter three numbers separated by spaces, then press the Enter key, or enter each number followed by a press of the Enter key

input, process, and output—called IPO

2.4 Identifiers

Identifiers are the names that identify the elements such as classes, methods, and variables in a program.

Ex. ComputeAverage, main, input, number1, number2, number3

All identifiers must obey the following rules:

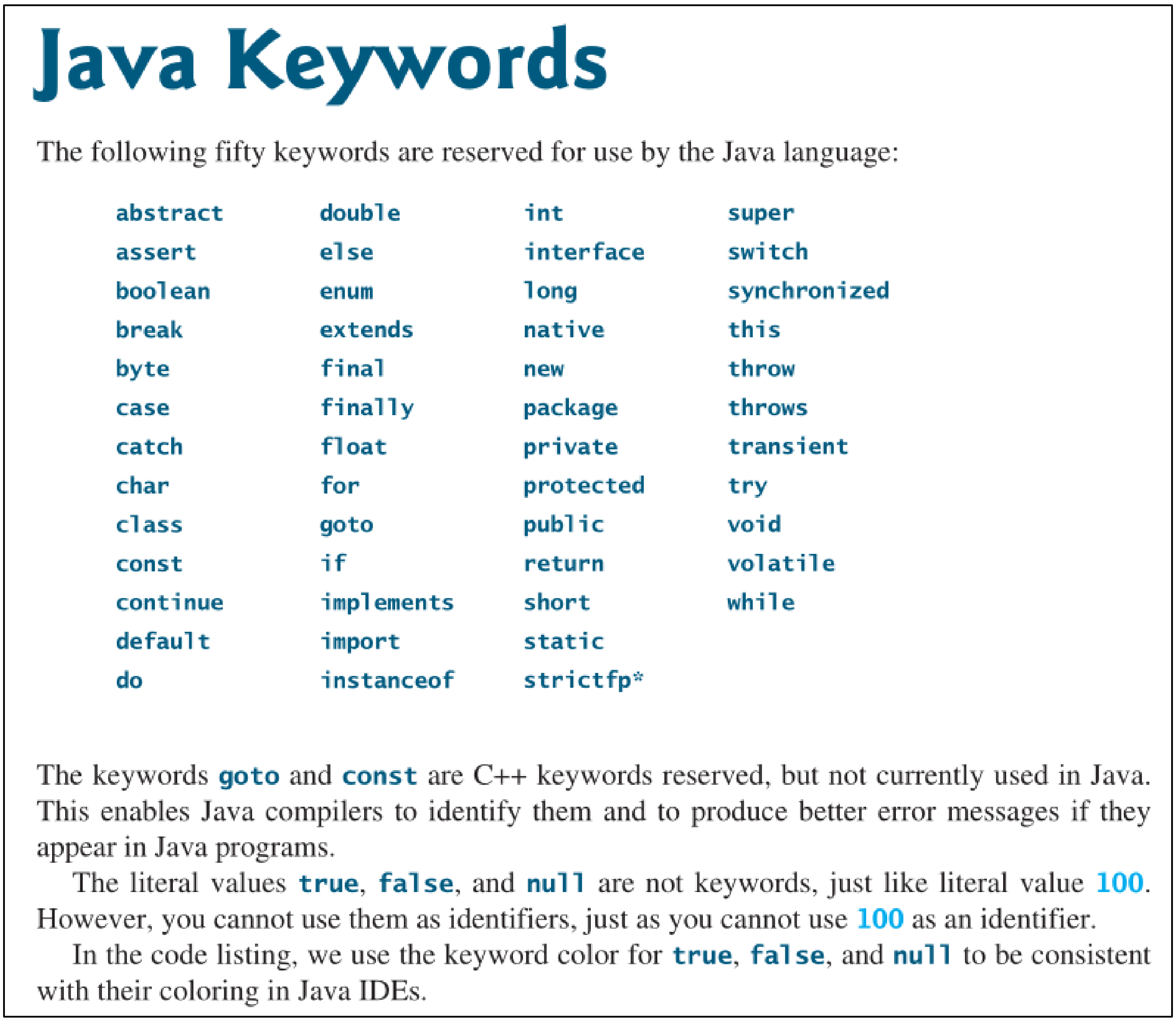
An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).

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.



Java Keywords

The following fifty keywords are reserved for use by the Java language:

abstract	double	int	super
assert	else	interface	switch
boolean	enum	long	synchronized
break	extends	native	this
byte	final	new	throw
case	finally	package	throws
catch	float	private	transient
char	for	protected	try
class	goto	public	void
const	if	return	volatile
continue	implements	short	while
default	import	static	
do	instanceof	strictfp*	

The keywords **goto** and **const** are C++ keywords reserved, but not currently used in Java. This enables Java compilers to identify them and to produce better error messages if they appear in Java programs.

The literal values **true**, **false**, and **null** are not keywords, just like literal value 100. However, you cannot use them as identifiers, just as you cannot use 100 as an identifier.

In the code listing, we use the keyword color for **true**, **false**, and **null** to be consistent with their coloring in Java IDEs.

The strictfp keyword is a modifier for a method or class that enables it to use strict floating-point calculations.

Since Java is case sensitive, area, Area, and AREA are all different identifiers.

Descriptive identifiers make programs easy to read. Avoid using abbreviations for identifiers.

Using complete words is more descriptive.

For example, numberOfStudents is better than numStuds, numOfStuds, or numOfStudents.

Do not name identifiers with the $ character. By convention, the $ character should be used only in mechanically generated source code.

2.5 Variables

Variables are used to represent values that may be changed in the program.

The variable declaration tells the compiler to allocate appropriate memory space for the variable based on its data type.

```
int count;             // Declare count to be an integer variable
double radius;         // Declare radius to be a double variable
double interestRate;   // Declare interestRate to be a double variable
```

data types int and double. Later you will be introduced to additional data types, such as byte, short, long, float, char, and boolean.

```
int i, j, k; // Declare i, j, and k as int variables
```

declare a variable and initialize it in one step

```
int i = 1, j = 2;
```

Whenever possible, declare a variable and assign its initial value in one step. This will make the program easy to read and avoid programming errors.

scope of a variable is the part of the program where the variable can be referenced.

2.6 Assignment Statements and Assignment

Expressions An assignment statement designates a value for a variable.

An assignment statement can be used as an expression in Java.

assignment operator equal sign (=)

char a;
a = 'A';
expression

```
int y = 1;                     // Assign 1 to variable y
double radius = 1.0;           // Assign 1.0 to variable radius
int x = 5 * (3 / 2);           // Assign the value of the expression to x
x = y + 1;                     // Assign the addition of y and 1 to x
double area = radius * radius * 3.14159; // Compute area
```

```
1 = x;   // Wrong
```

assignment expression

```
   System.out.println(x = 1);
```

which is equivalent to

```
   x = 1;
   System.out.println(x);
```

```
   i = j = k = 1;
```

which is equivalent to

```
   k = 1;
   j = k;
   i = j;
```

You cannot assign a double value (1.0) to an int variable without using type casting.

2.6 Identify and fix the errors in the following code:

```
public class Test {
  public static void main(String[] args) {
    int i = j = k = 2;
    System.out.println(i + " " + j + " " + k);
  }
}
```

2.7 Named Constants

A named constant is an identifier that represents a permanent value.

The value of a variable may change during the execution of a program, but a named constant, or simply constant, represents permanent data that never changes.

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

```
final double PI = 3.14159;
final int SIZE = 3;
```

A constant must be declared and initialized in the same statement.

The word final is a Java keyword for declaring a constant.

three benefits of using constants:

(1) you don't have to repeatedly type the same value if it is used multiple times;

(2) if you have to change the constant value (e.g., from 3.14 to 3.14159 for PI), you need to change it only in a single location in the source code; and

(3) a descriptive name for a constant makes the program easy to read.

2.8 Naming Conventions

Sticking with the Java naming conventions makes your programs easy to read and avoids errors.

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.

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

Do not choose class names that are already used in the 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

Java has six numeric types for integers and floating-point numbers with operators + , - , * , / , % .

2.9.1 Numeric Types

Every data type has a range of values.

The compiler allocates memory space for each variable or constant according to its data type.

Java provides eight primitive data types for numeric values, characters, and Boolean values.

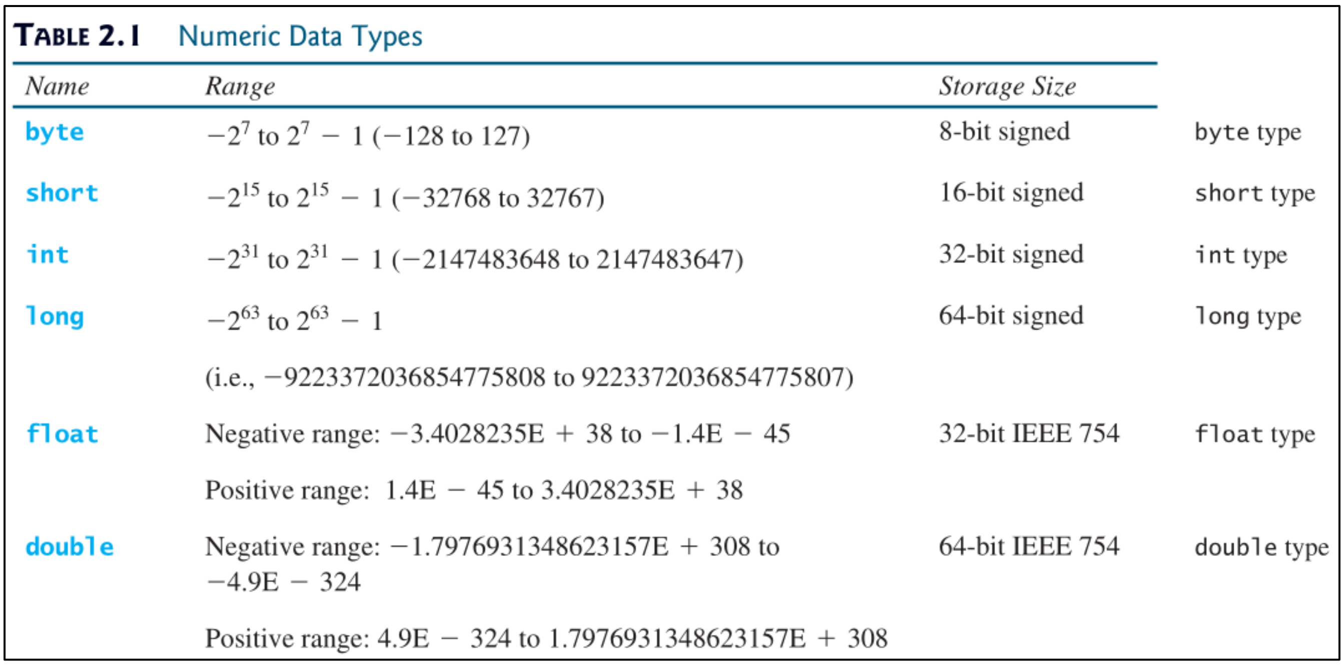


TABLE 2.1 Numeric Data Types

Name	*Range*	*Storage Size*	
`byte`	-2^7 to $2^7 - 1$ (-128 to 127)	8-bit signed	`byte` type
`short`	-2^{15} to $2^{15} - 1$ (-32768 to 32767)	16-bit signed	`short` type
`int`	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed	`int` type
`long`	-2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed	`long` type
`float`	Negative range: $-3.4028235\text{E} + 38$ to $-1.4\text{E} - 45$ Positive range: $1.4\text{E} - 45$ to $3.4028235\text{E} + 38$	32-bit IEEE 754	`float` type
`double`	Negative range: $-1.7976931348623157\text{E} + 308$ to $-4.9\text{E} - 324$ Positive range: $4.9\text{E} - 324$ to $1.7976931348623157\text{E} + 308$	64-bit IEEE 754	`double` type

four types for integers: byte, short, int, and long

Java uses two types for floating-point numbers: float and double.

The double type is twice as big as float, so the double is known as double precision and float as single precision.

2.9.2 Reading Numbers from the Keyboard

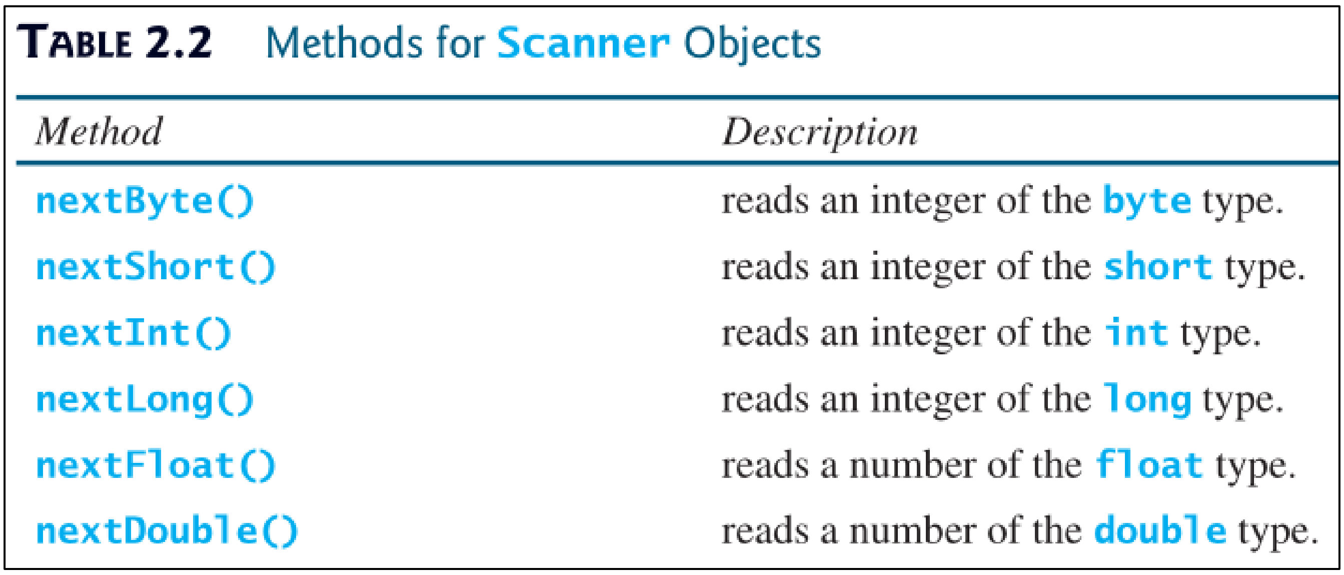


TABLE 2.2 Methods for `Scanner` Objects

Method	*Description*
`nextByte()`	reads an integer of the `byte` type.
`nextShort()`	reads an integer of the `short` type.
`nextInt()`	reads an integer of the `int` type.
`nextLong()`	reads an integer of the `long` type.
`nextFloat()`	reads a number of the `float` type.
`nextDouble()`	reads a number of the `double` type.

If you enter a value with an incorrect range or format, a runtime error would occur. For example, you enter a value 128 for line 3, an error would occur because 128 is out of range for a byte type integer.

2.9.3 Numeric Operators

The operators for numeric data types include the standard arithmetic operators: addition (+), subtraction (–), multiplication (*), division (/), and remainder (%)

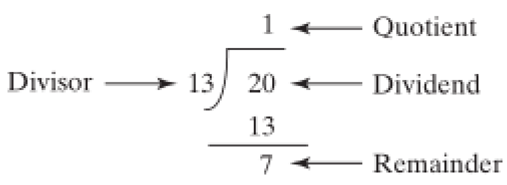
TABLE 2.3 Numeric Operators

Name	*Meaning*	*Example*	*Result*
+	Addition	34 + 1	35
-	Subtraction	34.0 – 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

For example, 5 / 2 yields 2, not 2.5, and –5 / 2 yields -2, not –2.5.

To perform a float-point division, one of the operands must be a floating-point number. For example, 5.0 / 2 yields 2.5.

20 % 13 yields 7



The remainder is negative only if the dividend is negative.

20 % -13 yields 7

-7 % 3 yields -1

-26 % -8 yields -2

The + and - operators can be both unary and binary.

A unary operator has only one operand; a binary operator has two.

For example, the - operator in -5 is a unary operator to negate number 5, whereas the - operator in 4 - 5 is a binary operator for subtracting 5 from 4.

2.9.4 Exponent Operations

Math.pow(a, b) method can be used to compute a^b.

```
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.

```java
int numberOfYears = 34;
double weight = 0.305;
```

2.10.1 Integer Literals

An integer literal can be assigned to an integer variable as long as it can fit into the variable.

The statement byte b = 128, for example, will cause a compile error, because 128 cannot be stored in a variable of the byte type. (Note that the range for a byte value is from –128 to 127.)

int type, whose value is between -2^{31} (-2147483648) and $2^{31} - 1$ (2147483647).

Long type, append the letter L or l to it. For example, to write integer 2147483648 in a Java program, you have to write it as 2147483648L or 2147483648l, because 2147483648 exceeds the range for the int value. L is preferred because l (lowercase L) can easily be confused with 1 (the digit one).

By default, an integer literal is a decimal integer number.

To denote a binary integer literal, use a leading 0b or 0B (zero B)

denote an octal integer literal, use a leading 0 (zero),

denote a hexadecimal integer literal, use a leading 0x or 0X (zero X).

```java
System.out.println(0B1111); // Displays 15
System.out.println(07777); // Displays 4095
System.out.println(0XFFFF); // Displays 65535
```

2.10.2 Floating-Point Literals

By default, a floating-point literal is treated as a double type value.

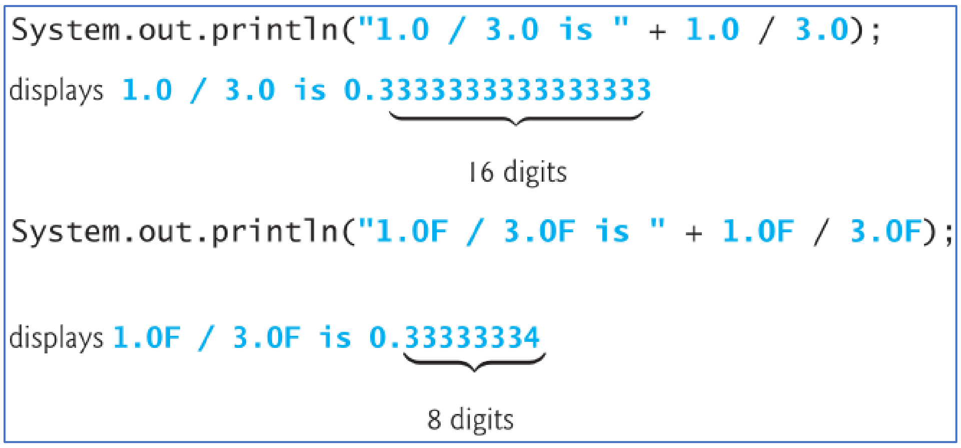
For example, 5.0 is considered a double value, not a float value.

You can make a number a float by appending the letter f or F

you can 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 the float type values.



```
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 Notation

Floating-point literals can be written in scientific notation in the form of $a * 10^b$.

123.456 is $1.23456 * 10^2$ and for 0.0123456 is $1.23456 * 10^{-2}$.

1.23456E2 or 1.23456E+2

1.23456 * 10-2 as 1.23456E-2. E (or e)

The float and double types are used to represent numbers with a decimal point.

Why are they called floating-point numbers?

These numbers are stored in scientific notation internally.

When a number such as 50.534 is converted into scientific notation, such as 5.0534E+1, its decimal point is moved (i.e., floated) to a new position.

To improve readability, Java allows you to use underscores between two digits in a number literal. For example, the following literals are correct.

```
long ssn = 232_45_4519;
long creditCardNumber = 2324_4545_4519_3415L;
```

2.11 Evaluating Expressions and Operator Precedence

Java expressions are evaluated in the same way as arithmetic expressions.

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.

When more than one operator is used in an expression, the following operator precedence rule is used to determine the order of evaluation.

Multiplication, division, and remainder operators are applied first.

If an expression contains several multiplication, division, and remainder operators, they are applied from left to right.

Addition and subtraction operators are applied last.

If an expression contains several addition and subtraction operators, they are applied from left to right.

2.13 Augmented Assignment Operators

The operators +, -, *, /, and % can be combined with the assignment operator to form augmented operators.

Table 2.4 Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Division assignment	`i /= 8`	`i = i / 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`

The augmented assignment operator is performed last after all the other operators in the expression are evaluated.

```
x /= 4 + 5.5 * 1.5;
```

is same as

```
x = x / (4 + 5.5 * 1.5);
```

There are no spaces in the augmented assignment operators. For example, + = should be +=.

```
x += 2; // Statement
System.out.println(x += 2); // Expression
```

2.14 Increment and Decrement Operators

The increment operator (++) and decrement operator (—) are for incrementing and decrementing a variable by 1.

These operators are known as postfix increment (or postincrement) and postfix decrement (or postdecrement), because the operators ++ and —— are placed after the variable.

```
int i = 3, j = 3;
i++; // i becomes 4
j--; // j becomes 2
```

These operators can also be placed before the variable.

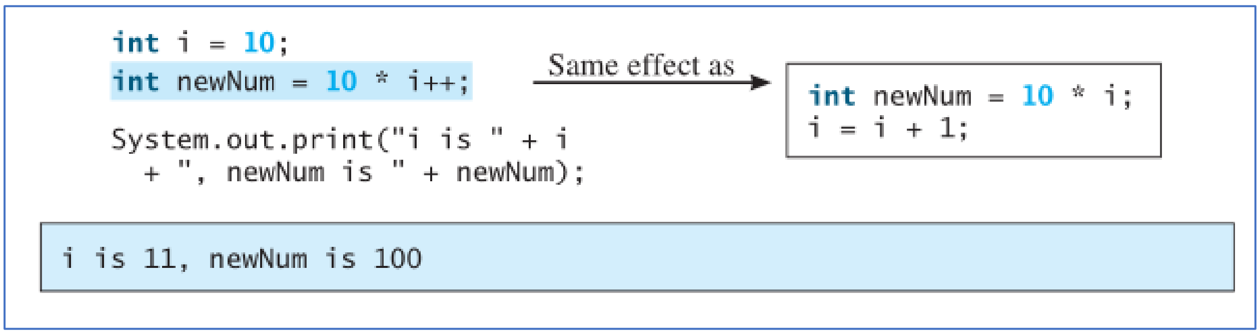
```
int i = 3, j = 3;
++i;  // i becomes 4
--j; // j becomes 2
```

++i increments i by 1 and ——j decrements j by 1.

These operators are known as prefix increment (or preincrement) and prefix decrement (or predecrement).

TABLE 2.5 Increment and Decrement Operators

Operator	*Name*	*Description*	*Example (assume i = 1)*
`++var`	preincrement	Increment `var` by `1`, and use the new `var` value in the statement	`int j = ++i;` `// j is 2, i is 2`
`var++`	postincrement	Increment `var` by `1`, but use the original `var` value in the statement	`int j = i++;` `// j is 1, i is 2`
`--var`	predecrement	Decrement `var` by `1`, and use the new `var` value in the statement	`int j = --i;` `// j is 0, i is 0`
`var--`	postdecrement	Decrement `var` by `1`, and use the original `var` value in the statement	`int j = i--;` `// j is 1, i is 0`

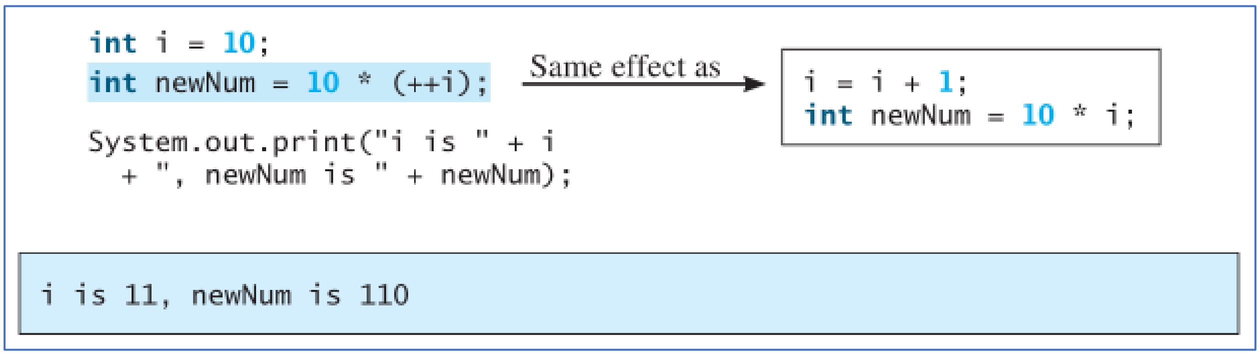


```
int i = 10;
int newNum = 10 * i++;

System.out.print("i is " + i
  + ", newNum is " + newNum);
```

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

```
i is 11, newNum is 100
```



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

System.out.print("i is " + i
  + ", newNum is " + newNum);
```

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

```
i is 11, newNum is 110
```

Avoid using these operators in expressions that modify multiple variables or the same variable multiple times, such as this one: int k = ++i + i.

2.15 Numeric Type Conversions

Floating-point numbers can be converted into integers using explicit casting.

Java automatically converts the integer to a floating-point value.

So, 3 * 4.5 is same as 3.0 * 4.5.

You can always assign a value to a numeric variable whose type supports a larger range of values; thus, for instance, you can assign a long value to a float variable.

You cannot, however, assign a value to a variable of a type with a smaller range unless you use type casting.

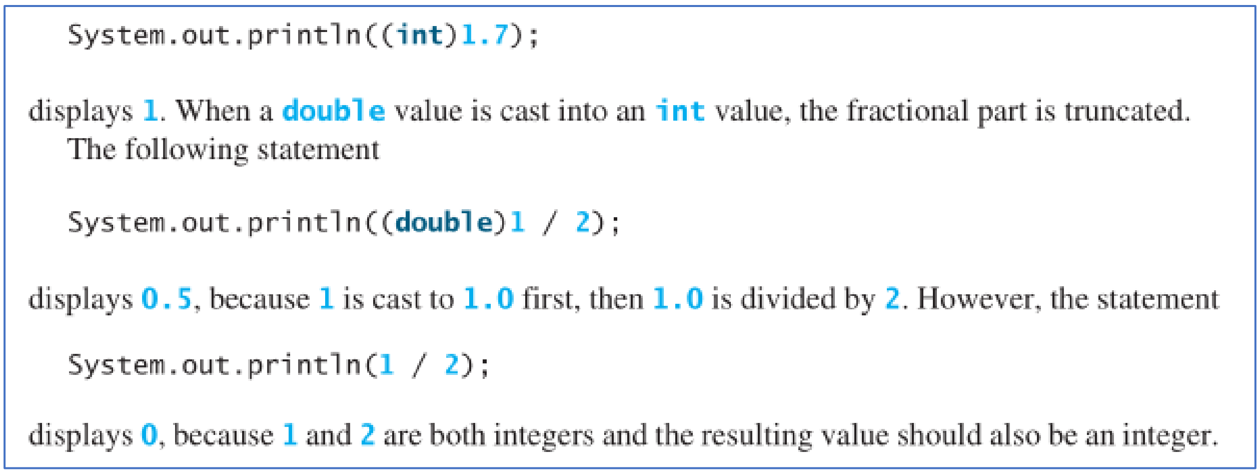
Casting is an operation that converts a value of one data type into a value of another data type.

Casting a type with a small range to a type with a larger range is known as widening a type.

Casting a type with a large range to a type with a smaller range is known as narrowing a type.

Java will automatically widen a type, but you must narrow a type explicitly.

syntax for casting a type is to specify the target type in parentheses, followed by the variable's name or the value to be cast.



```
System.out.println((int)1.7);
```

displays 1. When a double value is cast into an int value, the fractional part is truncated. The following statement

```
System.out.println((double)1 / 2);
```

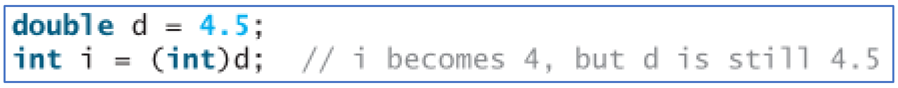
displays 0.5, because 1 is cast to 1.0 first, then 1.0 is divided by 2. However, the statement

```
System.out.println(1 / 2);
```

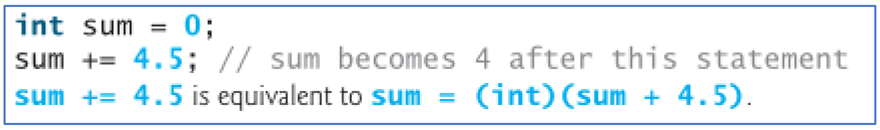
displays 0, because 1 and 2 are both integers and the resulting value should also be an integer.

Casting is necessary if you are assigning a value to a variable of a smaller type range, such as assigning a double value to an int variable. A compile error will occur if casting is not used in situations of this kind. However, be careful when using casting, as loss of information might lead to inaccurate results.

Casting does not change the variable being cast. For example, d is not changed after casting in the following code:

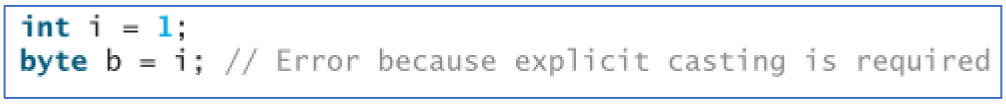


```
double d = 4.5;
int i = (int)d;   // i becomes 4, but d is still 4.5
```



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

sum += 4.5 is equivalent to sum = (int)(sum + 4.5).



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

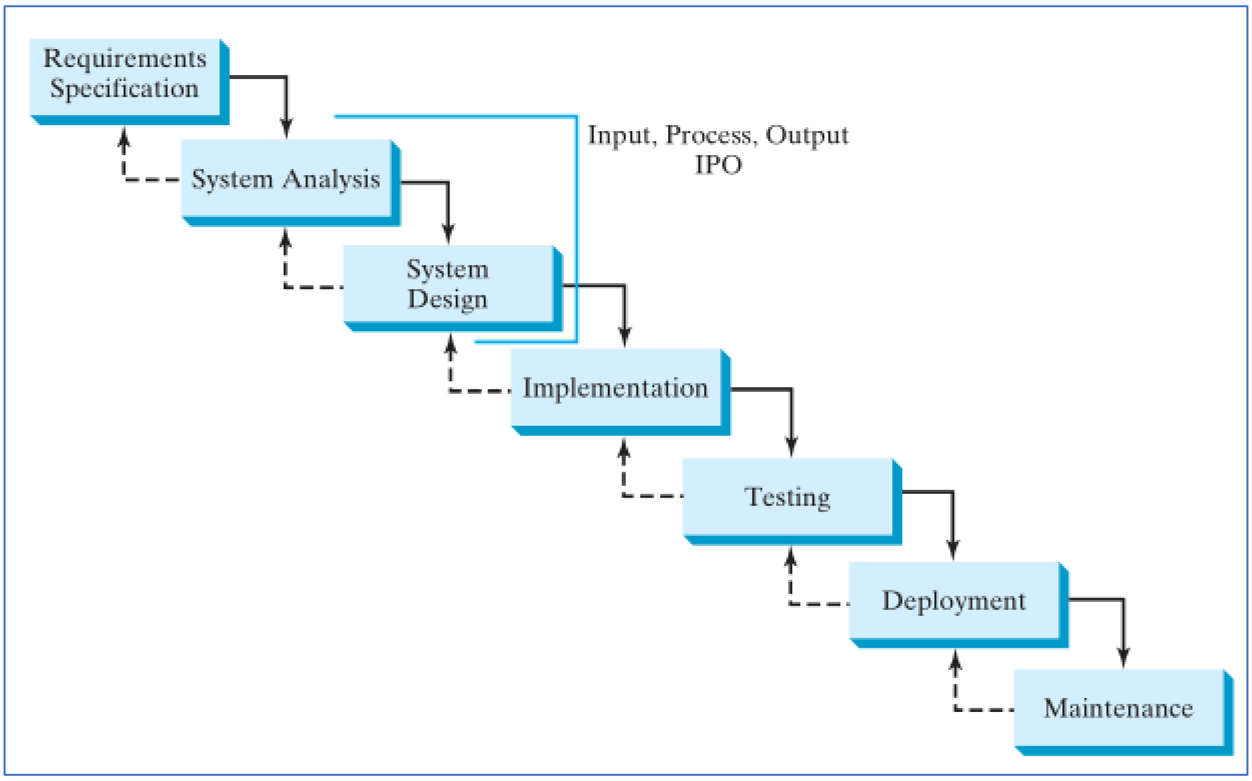
****

2.16 Software Development Process

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

Developing a software product is an engineering process.

Software products, no matter how large or how small, have the same life cycle.



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.

Requirements specification is a formal process that seeks to understand the problem that the software will address and to document in detail what the software system needs to do.

System analysis seeks to analyze the data flow and to identify the system's input and output.

System design is to design a process for obtaining the output from the input. input, process, and output (IPO).

Implementation involves translating the system design into programs.

Testing ensures that the code meets the requirements specification and weeds out bugs.

Deployment makes the software available for use.

Maintenance is concerned with updating and improving the product.

2.18 Common Errors and Pitfalls (Drawbacks,Difficulties)

Common elementary programming errors often involve undeclared variables, uninitialized variables, integer overflow, unintended integer division, and round-off errors.

Common Error 1: Undeclared/Uninitialized Variables and Unused Variables

Common Error 2: Integer Overflow

Numbers are stored with a limited numbers of digits.

When a variable is assigned a value that is too large (in size) to be stored, it causes overflow.

For example, executing the following statement causes overflow, because the largest value that can be stored in a variable of the int type is 2147483647. 2147483648 will be too large for an int value.

```
int value = 2147483647 + 1;
// value will actually be -2147483648
```

When a floating-point number is too small (i.e., too close to zero) to be stored, it causes underflow. Java approximates it to zero, so normally you don't need to be concerned about underflow.

Common Error 3: Round-off Errors

A round-off error, also called a rounding error, is the difference between the calculated approximation of a number and its exact mathematical value.

```
   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.

Common Error 4: Unintended Integer Division

Java uses the same divide operator, namely /, to perform both integer and floating-point division.

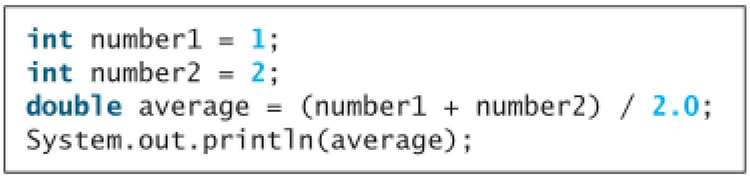
When two operands are integers, the / operator performs an integer division. The result of the operation is an integer. The fractional part is truncated.

To force two integers to perform a floating-point division, make one of the integers into a floating-point number.

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

(a)

(a)Output-1



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

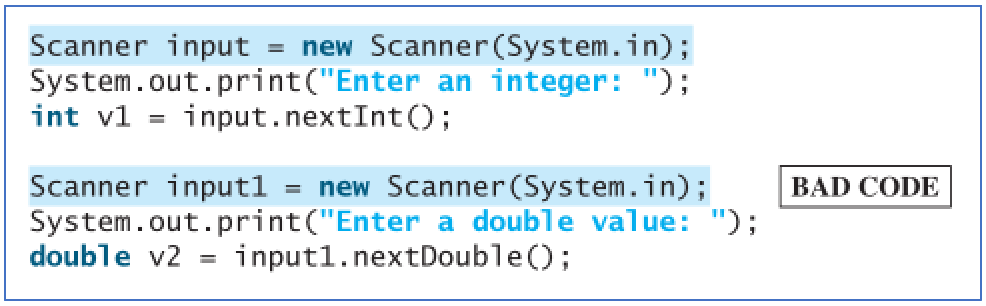
(b)

(b) Output- 1.5

Common Pitfall 1: Redundant Input Objects

New programmers often write the code to create multiple input objects for each input.

For example, the following code reads an integer and a double value.

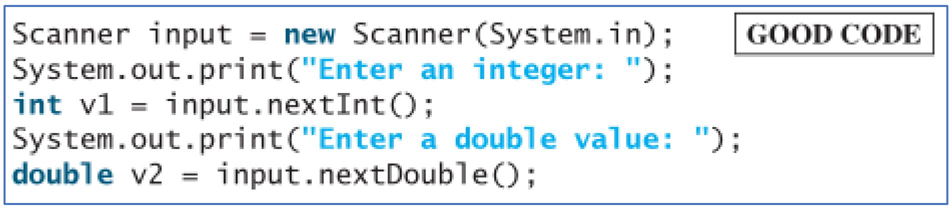


```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();

Scanner input1 = new Scanner(System.in);
System.out.print("Enter a double value: ");
double v2 = input1.nextDouble();
```

BAD CODE

The code is not wrong, but inefficient.



```
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();
```

GOOD CODE