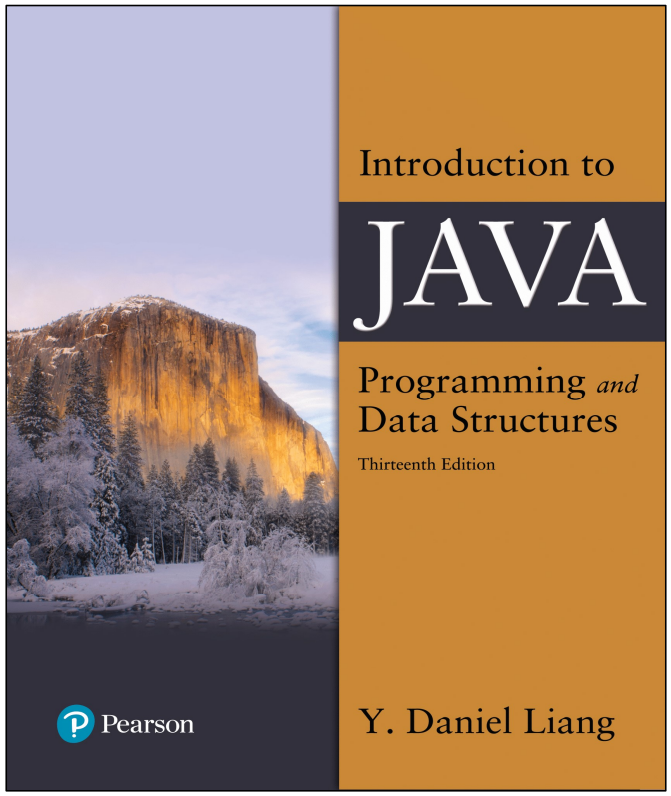
Introduction to Java Programming and Data Structures

Thirteenth Edition



Chapter 2

Elementary Programming

Pearson

Motivations

In the preceding chapter, you learned how to create, compile, and run a Java program. Starting from this chapter, you will learn how to solve practical problems programmatically. Through these problems, you will learn Java primitive data types and related subjects, such as variables, constants, data types, operators, expressions, and input and output.

Introducing Programming with an Example

Listing 2.1 Computing the Area of a Circle

➢Write a program to calculate the area of a circle with radius 20.

- Problem Solving
  - Analyze the problem
    - input/output
  - Design Solution
    - Algorithm
  - Coding
    - Program
  - Testing
    - Syntax error, run-time error, logical error

Algorithms

- Any computing problem can be solved by executing a series of actions in a specific order.
- An algorithm is a *procedure* for solving a problem in terms of
  - the actions to execute and
  - the order in which these actions execute
- Specifying the order in which statements (actions) execute in a program is called program control.

Write a program to calculate area of a circle with radius 20.

- Pseudocode is an informal language that helps you develop algorithms without having to worry about the strict details of Java language syntax.
- Flowchart uses diagrams and arrow lines to show flow of the statements.

Control Structures

- Any problem can be solved using 4 types of Control Structures
- Sequential execution: Statements in a program execute one after the other in the order in which they are written.
- Selection Structure: Enables you to specify which statement to execute based on a decision.
- iteration structure: Enables you to repeat execution of set of statements.
- Subprogram Structure: Enables you to break large problems to sub problems. By solving sub problems, you solve the entire problem

Sample Java Application Template

```
// This is a simple Java program.

public class Simple
{
      public static void main(String[] args)
      {

      }
}
```

This is the method header for the main method. The main method is where a Java application begins.

This area is the body of the main method. All of the actions to be completed during the main method will be between these curly braces.

Identifiers

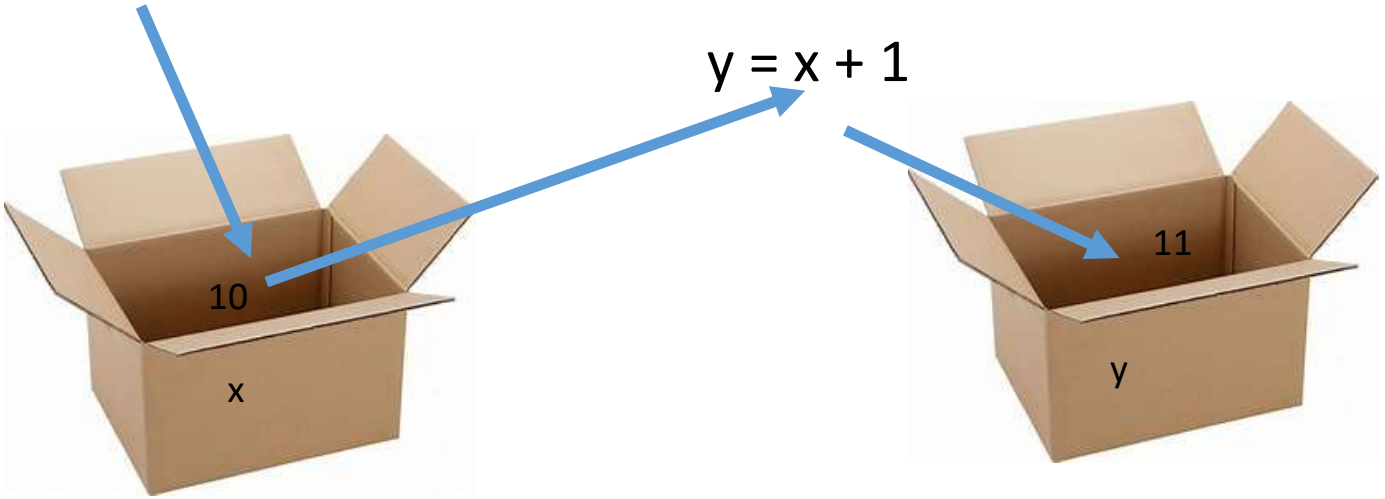
- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).
- An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
- An identifier cannot be a reserved word. (See Appendix A, “Java Keywords,” for a list of reserved words).
- An identifier cannot be `true`, `false`, or `null`.
- An identifier can be of any length.

Variables

- Related to variables in math
- A named “box” you can put a value in
- x = 10

same as in math:

y = x + 1



Memory Concepts
Variables

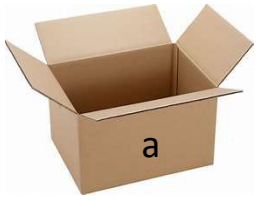
- Variables
  - Every variable has a name, a type, a size (in bytes) and a value.
  - When a new value is placed into a variable, the new value replaces the previous value (if any)
  - The previous value is lost, so this process is said to be *destructive*.
- Every variable must be declared before it can be used
- Syntax

  type name [= value];

Declaring Variables

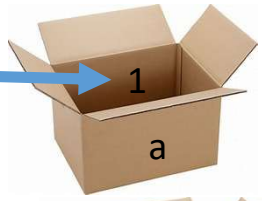
```
int x;                  // Declare x to be an
                        // integer variable;
double radius;          // Declare radius to
                        // be a double variable;

char a;                 // Declare a to be a
                        // character variable;
```



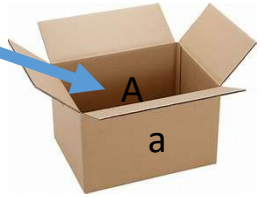
Assignment Statements

```
x = 1;                  // Assign 1 to x;
```




```
radius = 1.0;           // Assign 1.0 to radius;
```


```
Char a = 'A';           // Assign 'A' to a;
```



Declaring and Initializing in One Step

- `int x = 1;`
- `double d = 1.4;`

Named Constants

```
final datatype CONSTANTNAME = VALUE;
final double PI = 3.14159;
final int SIZE = 3;
```

Naming Conventions (1 of 2)

- Choose meaningful and descriptive names.
- Variables and method names:
  - Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name. For example, the variables `radius` and `area`, and the method `computeArea`.

Naming Conventions (2 of 2)

- 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`

Numerical Data Types

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

Opening Problem

Listing 2.1 Computing the Area of a Circle

This program computes the area of the circle with radius 20.

ComputeArea

1) radius = 20
2) area = radius * radius * 3.14159
3) Print area

Reading Numbers from the Keyboard

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

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.

Opening Problem (interactive)

This program computes the area of the circle with a given radius.

1) Input radius
2) area = radius * radius * 3.14159
3) Print area

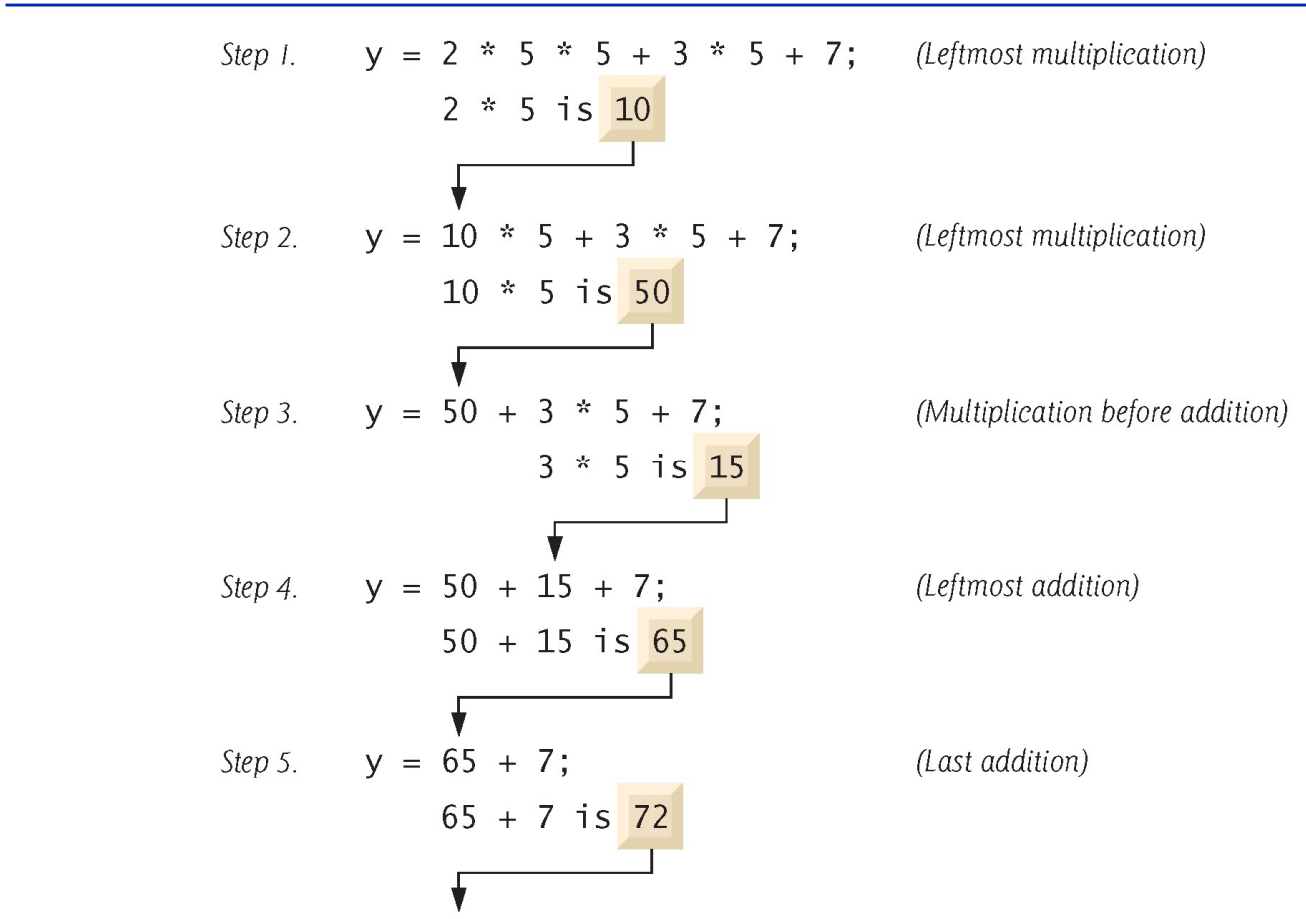
Calculate Gross Pay

- Given hours worked and pay rate, calculate and print gross pay.

1) Analyze the problem (input/output)
2) Algorithm
3) Coding
4) Testing

Arithmetic Operators

Operator(s)	Operation(s)	Order of evaluation (precedence)
*	Multiplication	Evaluated first. If there are several operators of this type, they're evaluated from *left to right*.
/	Division	
%	Remainder	
+	Addition	Evaluated next. If there are several operators of this type, they're evaluated from *left to right*.
-	Subtraction	
=	Assignment	Evaluated last.


Step 1.
y = 2 * 5 * 5 + 3 * 5 + 7;
2 * 5 is 10
(Leftmost multiplication)
Step 2.
y = 10 * 5 + 3 * 5 + 7;
10 * 5 is 50
(Leftmost multiplication)
Step 3.
y = 50 + 3 * 5 + 7;
3 * 5 is 15
(Multiplication before addition)
Step 4.
y = 50 + 15 + 7;
50 + 15 is 65
(Leftmost addition)
Step 5.
y = 65 + 7;
65 + 7 is 72
(Last addition)
Step 6.
y = 72
(Last operation—place 72 in y)

NOTE

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

System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);

displays 0.5000000000000001, not 0.5, and

System.out.println(1.0 - 0.9);

displays 0.09999999999999998, not 0.1. Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.

Number Literals

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

```
int i = 34;

long x = 1000000;

double d = 5.0;
```

Arithmetic Expressions

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

is translated to

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

Problem: Converting Temperatures

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

$$celsius = (\tfrac{5}{9})(fahrenheit - 32)$$

Note: you have to write
celsius = (5.0 / 9) * (fahrenheit – 32)

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`

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`

Increment and Decrement Operators, cont.

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

Same effect as

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

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

Same effect as

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

Numeric Type Conversion

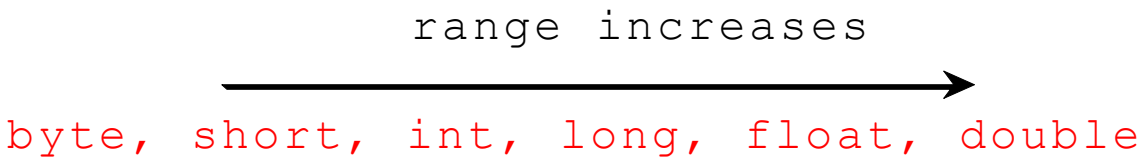
Consider the following statements:

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

Conversion Rules

When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:

1. If one of the operands is double, the other is converted into double.
2. Otherwise, if one of the operands is float, the other is converted into float.
3. Otherwise, if one of the operands is long, the other is converted into long.
4. Otherwise, both operands are converted into int.



Type Casting

Implicit casting

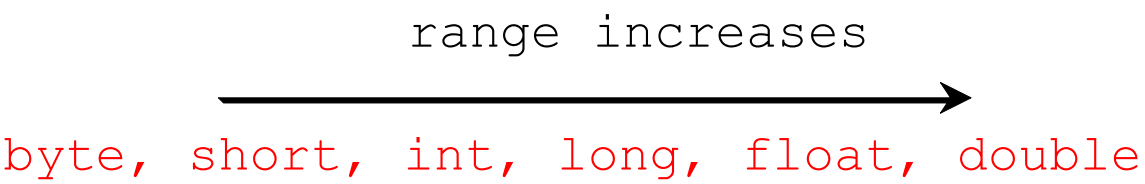
```
double d = 3;
```
(type widening)

Explicit casting

```
int i = (int)3.0;
```
(type narrowing)

```
int i = (int)3.9;
```
(Fraction part is truncated)

What is wrong? int x = 5 / 2.0;



Casting in an Augmented Expression

In Java, an augmented expression of the form **`x1 op= x2`** is implemented as **`x1 = (T)(x1 op x2)`**, where **`T`** is the type for **`x1`**. Therefore, the following code is correct.

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

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

Problem: Keeping Two Digits After Decimal Points

Write a program that displays the sales tax with two digits after the decimal point.

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.

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

Problem: 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.

Common Error 1: Undeclared/Uninitialized Variables and Unused Variables

```
double interestRate = 0.05;
double interest = interestrate * 45;
```

Common Error 2: Integer Overflow

```
int value = 2147483647 + 1;

// value will actually be     –2147483648
```

Common Error 3: Round-off Errors

```
System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
System.out.println(1.0 - 0.9);
```

Common Error 4: Unintended Integer Division

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

(a)

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

(b)

Common Pitfall 1: Redundant Input Objects

```
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();
```