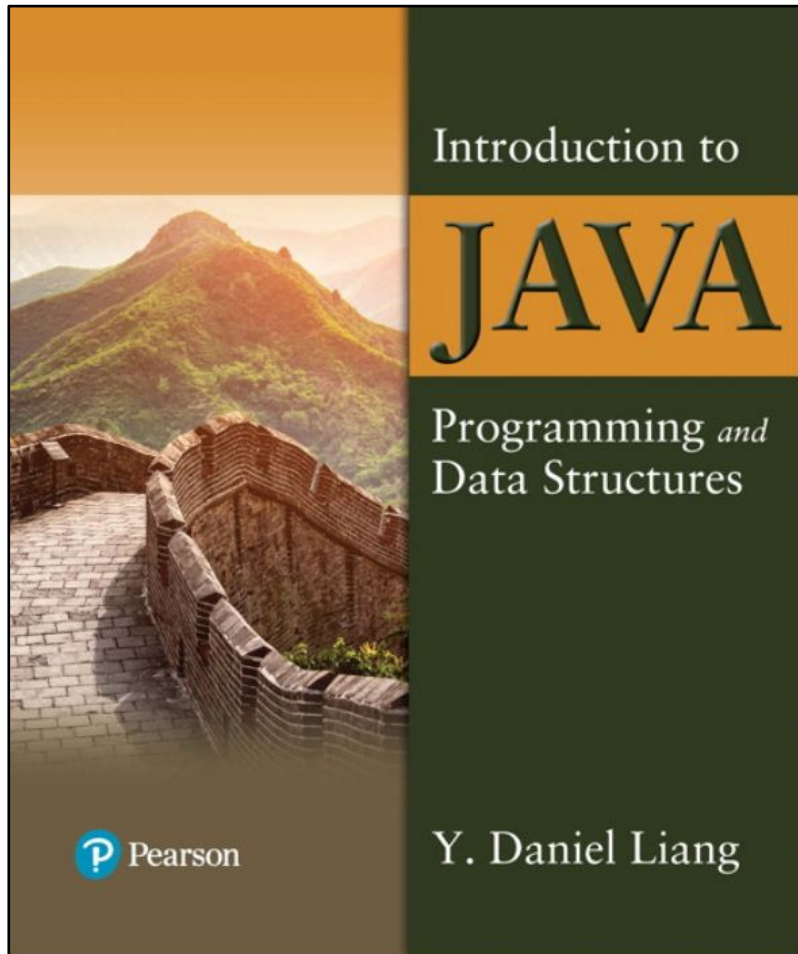


Introduction to Java Programming and Data Structures

Twelfth Edition



Chapter 2

Elementary Programming

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.

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

Declaring and Initializing in One Step

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

Introducing Programming With an Example

Listing 2.1 Computing the Area of a Circle

This program computes the area of the circle.

[ComputeArea](#)

Note: Clicking the green button displays the source code with interactive animation. You can also run the code in a browser. Internet connection is needed for this button.

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

Reading Input From the Console

1. Create a Scanner object

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

2. Use the method `nextDouble()` to obtain to a double value. For example,

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

[ComputeAreaWithConsoleInput](#)

[ComputeAverage](#)

Reading Numbers From the Keyboard

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

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

Implicit Import and Explicit Import

```
java.util.* ; // Implicit import
```

```
java.util.Scanner; // Explicit Import
```

No performance difference

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

Integer Division

+, −, *, /, and %

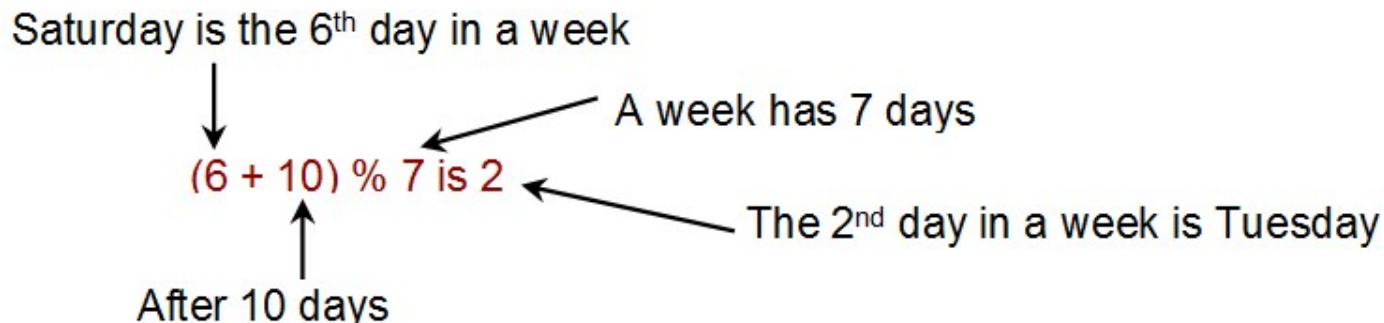
$5 / 2$ yields an integer 2.

$5.0 / 2$ yields a double value 2.5

$5 \% 2$ yields 1 (the remainder of the division)

Remainder Operator

Remainder is very useful in programming. For example, an even number $\% 2$ is always 0 and an odd number $\% 2$ is always 1. So you can use this property to determine whether a number is even or odd. Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:



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 0.5000000000000000001, not 0.5, and

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

displays 0.0999999999999999998, not 0.1. Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.

Exponent Operations

```
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
```

double vs float

The double type values are more accurate than the float type values. For example,

```
System.out.println("1.0 / 3.0 is " + 1.0 /  
3.0);
```

displays 1.0 / 3.0 is 0.3333333333333333
16 digits

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

displays 1.0F / 3.0F is 0.33333334
7 digits

Scientific Notation

Floating-point literals can also be specified in scientific notation, for example, $1.23456e+2$, same as $1.23456e2$, is equivalent to 123.456 , and $1.23456e-2$ is equivalent to 0.0123456 . E (or e) represents an exponent and it can be either in lowercase or uppercase.

Problem: Converting Temperatures

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

$$celsius = \left(\frac{5}{9}\right)(fahrenheit - 32)$$

Note: you have to write

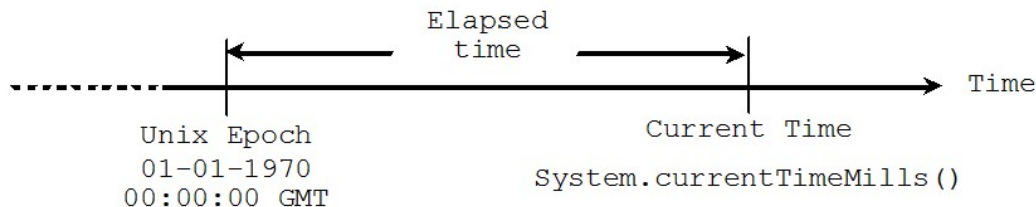
$$celsius = (5.0 / 9) * (fahrenheit - 32)$$

[FahrenheitToCelsius](#)

Problem: Displaying Current Time

Write a program that displays current time in GMT 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.



[ShowCurrentTime](#)

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 (1 of 3)

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

Increment and Decrement Operators (2 of 3)

```
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;
```

Increment and Decrement Operators (3 of 3)

Using increment and decrement operators makes expressions short, but it also makes them complex and difficult to read. Avoid using these operators in expressions that modify multiple variables, or the same variable for multiple times such as this: `int k = ++i + 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;`

range increases
→
byte, short, int, long, float, double

Copyright



This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.