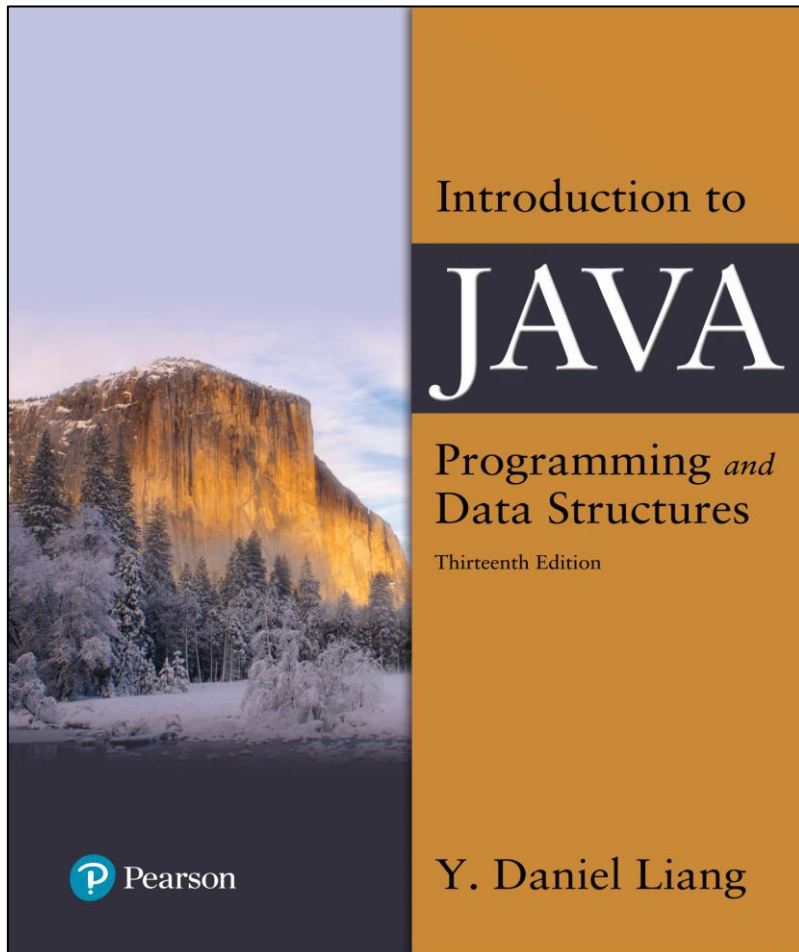


Introduction to Java Programming and Data Structures

Thirteenth Edition



Chapter 4

Mathematical Functions,
Characters, and Strings

Mathematical Functions

Java provides many useful methods in the **Math** class for performing common mathematical functions.

The Math Class

- Class constants:
 - `PI`
 - `E`
- Class methods:
 - Trigonometric Methods
 - Exponent Methods
 - Rounding Methods
 - `min`, `max`, `abs`, **and** `random` **Methods**

Trigonometric Methods

- `sin(double a)`
- `cos(double a)`
- `tan(double a)`
- `acos(double a)`
- `asin(double a)`
- `atan(double a)`

Radians

`toRadians(90)`

Examples:

`Math.sin(0)` returns 0.0

`Math.sin(Math.PI / 6)`
returns 0.5

`Math.sin(Math.PI / 2)`
returns 1.0

`Math.cos(0)` returns 1.0

`Math.cos(Math.PI / 6)`
returns 0.866

`Math.cos(Math.PI / 2)`
returns 0

Exponent Methods

- **`exp(double a)`**
Returns e raised to the power of a .
- **`log(double a)`**
Returns the natural logarithm of a .
- **`log10(double a)`**
Returns the 10-based logarithm of a .
- **`pow(double a, double b)`**
Returns a raised to the power of b .
- **`sqrt(double a)`**
Returns the square root of a .

Examples:

`Math.exp(1)` returns 2.71

`Math.log(2.71)` returns 1.0

`Math.pow(2, 3)` returns 8.0

`Math.pow(3, 2)` returns 9.0

`Math.pow(3.5, 2.5)` returns 22.91765

`Math.sqrt(4)` returns 2.0

`Math.sqrt(10.5)` returns 3.24

Rounding Methods

- **`double ceil(double x)`**
`x` rounded up to its nearest integer. This integer is returned as a double value.
- **`double floor(double x)`**
`x` is rounded down to its nearest integer. This integer is returned as a double value.
- **`double rint(double x)`**
`x` is rounded to its nearest integer. If `x` is equally close to two integers, the even one is returned as a double.
- **`int round(float x)`**
Return `(int)Math.floor(x+0.5)` .
- **`long round(double x)`**
Return `(long)Math.floor(x+0.5)` .

Rounding Methods Examples (1 of 2)

`Math.ceil(2.1)` returns 3.0

`Math.ceil(2.0)` returns 2.0

`Math.ceil(-2.0)` returns -2.0

`Math.ceil(-2.1)` returns -2.0

`Math.floor(2.1)` returns 2.0

`Math.floor(2.0)` returns 2.0

`Math.floor(-2.0)` returns -2.0

`Math.floor(-2.1)` returns -3.0

`Math rint(2.1)` returns 2.0

`Math.rint(2.0)` returns 2.0

`Math.rint(-2.0)` returns -2.0

`Math.rint(-2.1)` returns -2.0

`Math.rint(2.5)` returns 2.0

Rounding Methods Examples (2 of 2)

`Math rint(-2.5)` returns `-2.0`

`Math.round(2.6f)` returns `3`

`Math.round(2.0)` returns `2`

`Math.round(-2.0f)` returns `-2`

`Math.round(-2.6)` returns `-3`

min, max, and abs

- `max(a, b)` and `min(a, b)`

Returns the maximum or minimum of two parameters.

- `abs(a)`

Returns the absolute value of the parameter.

- `random()`

Returns a random double value in the range [0.0, 1.0).

Examples:

`Math.max(2, 3)` returns 3

**`Math.max(2.5, 3)` returns
3.0**

**`Math.min(2.5, 3.6)`
returns 2.5**

`Math.abs(-2)` returns 2

**`Math.abs(-2.1)` returns
2.1**

The random Method

Generates a random **double** value greater than or equal to 0.0 and less than 1.0 ($0 \leq \text{Math.random}() < 1.0$).

Examples:

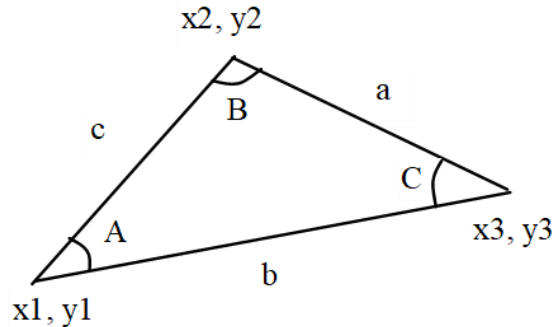
`(int) (Math.random() * 10)` \longrightarrow Returns a random integer between 0 and 9.

`50 + (int) (Math.random() * 50)` \longrightarrow Returns a random integer between 50 and 99.

In general,

`a + Math.random() * b` \longrightarrow Returns a random number between a and a + b, excluding a + b.

Case Study: Computing Angles of a Triangle



$$A = \arccos \left(\frac{a^2 - b^2 - c^2}{-2bc} \right)$$
$$B = \arccos \left(\frac{b^2 - a^2 - c^2}{-2ac} \right)$$
$$C = \arccos \left(\frac{c^2 - b^2 - a^2}{-2ab} \right)$$

Write a program that prompts the user to enter the x- and y-coordinates of the three corner points in a triangle and then displays the triangle's angles.

[ComputeAngles](#)

Character Data Type

```
char letter = 'A'; (ASCII)
```

```
char numChar = '4'; (ASCII)
```

```
char letter = '\u0041'; (Unicode)
```

```
char numChar = '\u0034'; (Unicode)
```

Four hexadecimal digits.

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**.

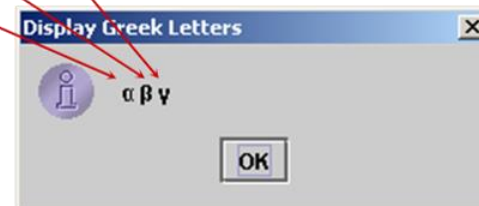
```
char ch = 'a';
```

```
System.out.println(++ch);
```

Unicode Format

Java characters use **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. Unicode takes two bytes, preceded by `\u`, expressed in four hexadecimal numbers that run from `'\u0000'` to `'\uFFFF'`. So, Unicode can represent 65535 + 1 characters.

Unicode `\u03b1` `\u03b2` `\u03b3` for three
Greek letters



ASCII Code for Commonly Used Characters

Characters	Code Value in Decimal	Unicode Value
'0' to '9'	48 to 57	\u0030 to \u0039
'A' to 'Z'	65 to 90	\u0041 to \u005A
'a' to 'z'	97 to 122	\u0061 to \u007A

Escape Sequences for Special Characters

Escape Sequence	Name	Unicode Code	Decimal Value
<code>\b</code>	Backspace	<code>\u0008</code>	8
<code>\t</code>	Tab	<code>\u0009</code>	9
<code>\n</code>	Linefeed	<code>\u000A</code>	10
<code>\f</code>	Formfeed	<code>\u000C</code>	12
<code>\r</code>	Carriage Return	<code>\u000D</code>	13
<code>\\</code>	Backslash	<code>\u005C</code>	92
<code>\"</code>	Double Quote	<code>\u0022</code>	34

Appendix B: ASCII Character Set (1 of 2)

ASCII Character Set is a subset of the Unicode from \u0000 to \u007f

Table B.1 ASCII Character Set in the Decimal Index

	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
1	nl	vt	ff	cr	so	si	dle	dcl	dc2	dc3
2	dc4	nak	syn	etb	can	em	sub	esc	fs	gs
3	rs	us	sp	!	“	#	\$	%	&	‘
4	()	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[\]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{		}	~	del	-	-

Appendix B: ASCII Character Set (2 of 2)

ASCII Character Set is a subset of the Unicode from \u0000 to \u007f

Table B.2 ASCII Character Set in the Hexadecimal Index

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht	nl	vt	ff	cr	so	si
1	dle	dcl	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
2	sp	!	“	#	\$	%	&	‘	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	del

Casting Between char and Numeric Types

```
int i = 'a'; // Same as int i = (int) 'a';
```

```
char c = 97; // Same as char c = (char) 97;
```

Comparing and Testing Characters

```
if (ch >= 'A' && ch <= 'Z')
```

```
    System.out.println(ch + " is an uppercase letter");
```

```
else if (ch >= 'a' && ch <= 'z')
```

```
    System.out.println(ch + " is a lowercase letter");
```

```
else if (ch >= '0' && ch <= '9')
```

```
    System.out.println(ch + " is a numeric character");
```

Methods in the Character Class

Method	Description
<code>isDigit(ch)</code>	Returns true if the specified character is a digit.
<code>isLetter(ch)</code>	Returns true if the specified character is a letter.
<code>isLetterOfDigit(ch)</code>	Returns true if the specified character is a letter or digit.
<code>isLowerCase(ch)</code>	Returns true if the specified character is a lowercase letter.
<code>isUpperCase(ch)</code>	Returns true if the specified character is an uppercase letter.
<code>toLowerCase(ch)</code>	Returns the lowercase of the specified character.
<code>toUpperCase(ch)</code>	Returns the uppercase of the specified character.

The String Type

The char type only represents one character. To represent a string of characters, use the data type called String. For example,

```
String message = "Welcome to Java";
```

String is actually a predefined class in the Java library just like the System class and Scanner 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 9, “Objects and Classes.” For the time being, you just need to know how to declare a String variable, how to assign a string to the variable, how to concatenate strings, and to perform simple operations for strings.

Simple Methods for String Objects (1 of 2)

Method	Description
<code>length()</code>	Returns the number of characters in this string.
<code>charAt(index)</code>	Returns the character at the specified index from this string.
<code>concat(s1)</code>	Returns a new string that concatenates this string with string <code>s1</code> .
<code>toUpperCase()</code>	Returns a new string with all letters in uppercase.
<code>toLowerCase()</code>	Returns a new string with all letters in lowercase.
<code>trim()</code>	Returns a new string with whitespace characters trimmed on both sides.

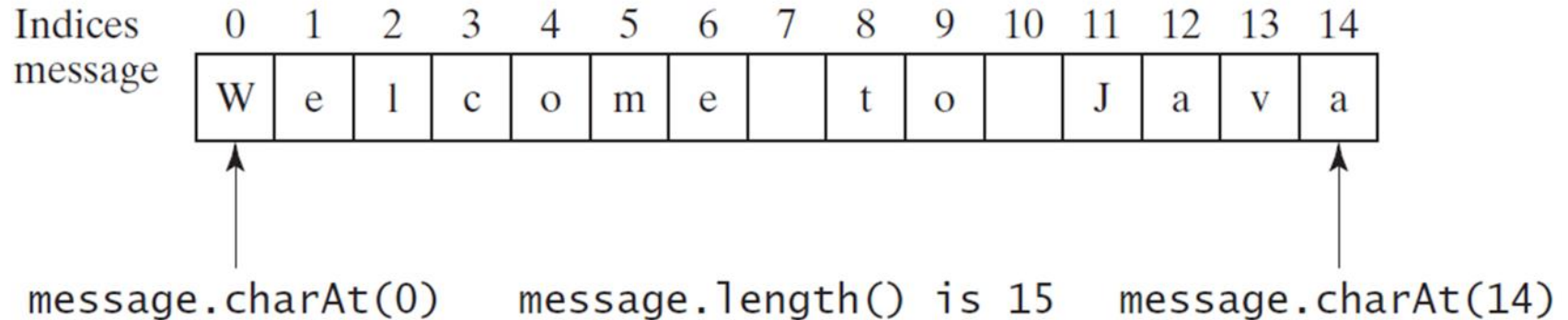
Simple Methods for String Objects (2 of 2)

Strings are objects in Java. The methods in the preceding table can only be invoked from a specific string instance. For this reason, these methods are called **instance methods**. A non-instance method is called a **static method**. A static method can be invoked without using an object. All the methods defined in the **Math** class are static methods. They are not tied to a specific object instance. The syntax to invoke an instance method is **referenceVariable.methodName(arguments)**.

Getting String Length

```
String message = "Welcome to Java";  
  
System.out.println("The length of " + message + " is "  
    + message.length());
```


Getting Characters From a String



```
String message = "Welcome to Java";
```

```
System.out.println("The first character in message is "  
+ message.charAt(0) );
```

Converting Strings

`"Welcome".toLowerCase()` returns a new string, `welcome`.

`"Welcome".toUpperCase()` returns a new string, `WELCOME`.

`" Welcome ".trim()` returns a new string, `Welcome`.

String Concatenation

```
String s3 = s1.concat(s2); or String s3 = s1 + s2;
```

```
// Three strings are concatenated
```

```
String message = "Welcome " + "to " + "Java";
```

```
// 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'; // s1 becomes SupplementB
```

Reading a String From the Console

```
Scanner input = new Scanner(System.in);  
  
System.out.print("Enter three words  
separated by spaces: ");  
  
String s1 = input.next();  
String s2 = input.next();  
String s3 = input.next();  
  
System.out.println("s1 is " + s1);  
System.out.println("s2 is " + s2);  
System.out.println("s3 is " + s3);
```

Reading a Character From the Console

```
Scanner input = new Scanner(System.in);  
  
System.out.print("Enter a character: ");  
  
String s = input.nextLine();  
  
char ch = s.charAt(0);  
  
System.out.println("The character entered is " + ch);
```

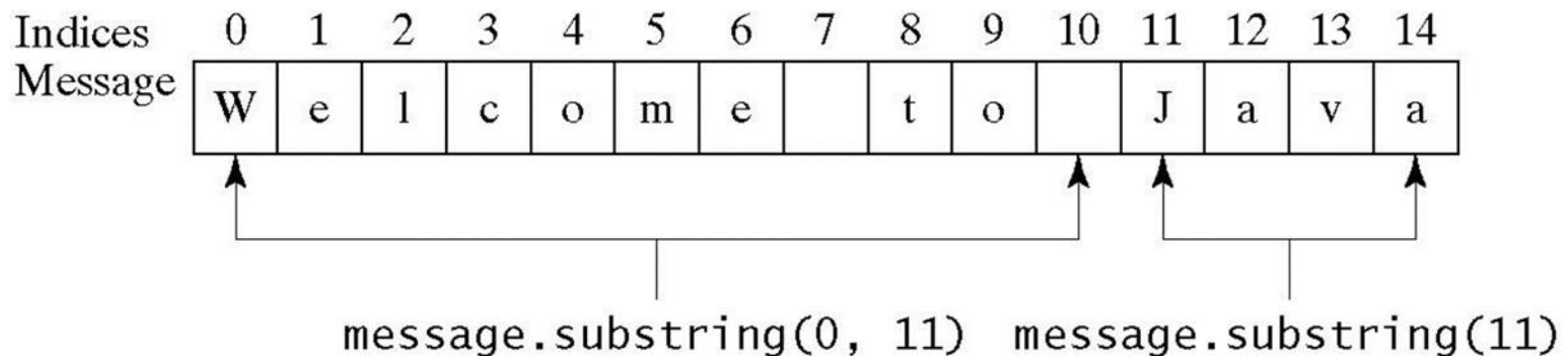
Comparing Strings

Method	Description
<code>equals(s1)</code>	Returns true if this string is equal to string <code>s1</code> .
<code>equalsIgnoreCase(s1)</code>	Returns true if this string is equal to string <code>s1</code> ; it is case insensitive.
<code>compareTo(s1)</code>	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than <code>s1</code> .
<code>compareToIgnoreCase(s1)</code>	Same as <code>compareTo</code> except that the comparison is case insensitive.
<code>startsWith(prefix)</code>	Returns true if this string starts with the specified prefix.
<code>endsWith(suffix)</code>	Returns true if this string ends with the specified suffix.

[OrderTwoCities](#)

Obtaining Substrings

Method	Description
<code>substring(beginIndex)</code>	Returns this string's substring that begins with the character at the specified <code>beginIndex</code> and extends to the end of the string, as shown in Figure 4.2.
<code>substring(beginIndex, endIndex)</code>	Returns this string's substring that begins at the specified <code>beginIndex</code> and extends to the character at index <code>endIndex - 1</code> , as shown in Figure 9.6. Note that the character at <code>endIndex</code> is not part of the substring.



Finding a Character or a Substring in a String (1 of 2)

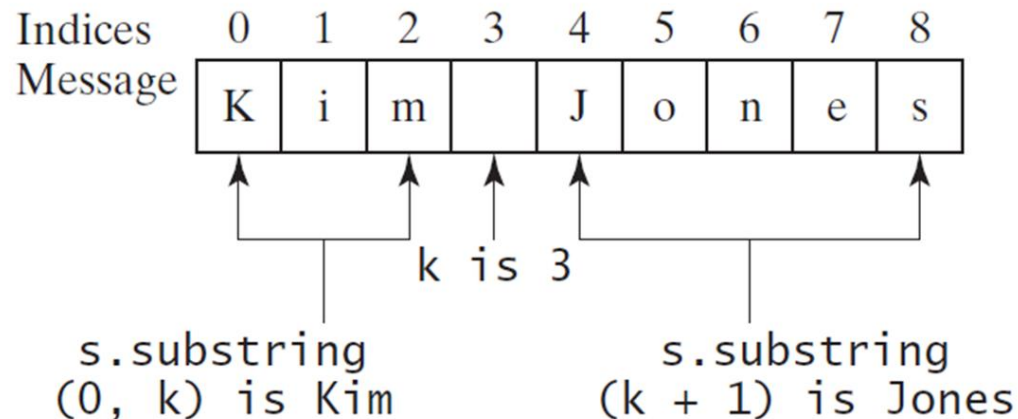
Method	Description
<code>indexOf(ch)</code>	Returns the index of the first occurrence of <code>ch</code> in the string. Returns <code>-1</code> if not matched.
<code>indexOf(ch, fromIndex)</code>	Returns the index of the first occurrence of <code>ch</code> after <code>fromIndex</code> in the string. Returns <code>-1</code> if not matched.
<code>indexOf(s)</code>	Returns the index of the first occurrence of string <code>s</code> in this string. Returns <code>-1</code> if not matched.
<code>indexOf(s, fromIndex)</code>	Returns the index of the first occurrence of string <code>s</code> in this string after <code>fromIndex</code> . Returns <code>-1</code> if not matched.
<code>lastIndexOf(ch)</code>	Returns the index of the last occurrence of <code>ch</code> in the string. Returns <code>-1</code> if not matched.
<code>lastIndexOf(ch, fromIndex)</code>	Returns the index of the last occurrence of <code>ch</code> before <code>fromIndex</code> in this string. Returns <code>-1</code> if not matched.
<code>lastIndexOf(s)</code>	Returns the index of the last occurrence of string <code>s</code> . Returns <code>-1</code> if not matched.
<code>lastIndexOf(s, fromIndex)</code>	Returns the index of the last occurrence of string <code>s</code> before <code>fromIndex</code> . Returns <code>-1</code> if not matched.

Finding a Character or a Substring in a String (2 of 2)

```
int k = s.indexOf(' ');
```

```
String firstName = s.substring(0, k);
```

```
String lastName = s.substring(k + 1);
```



Conversion Between Strings and Numbers

```
int intValue = Integer.parseInt(intString);
```

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

```
String s = number + "";
```

Case Study: Converting a Hexadecimal Digit to a Decimal Value

Write a program that converts a hexadecimal digit into a decimal value.

[HexDigit2Dec](#)

Case Study: Revising the Lottery Program Using Strings

A problem can be solved using many different approaches. This section rewrites the lottery program in Listing 3.7 using strings. Using strings simplifies this program.

[LotteryUsingStrings](#)

Formatting Output

Use the `printf` statement.

```
System.out.printf(format, items);
```

Where `format` is a string that may consist of substrings and format specifiers. A format specifier specifies how an item should be displayed. An item may be a numeric value, character, boolean value, or a string. Each specifier begins with a percent sign.

Frequently-Used Specifiers

Specifier	Output	Example
<code>%b</code>	a boolean value	true or false
<code>%c</code>	a character	'a'
<code>%d</code>	a decimal integer	200
<code>%f</code>	a floating-point number	45.460000
<code>%e</code>	a number in standard scientific notation	4.556000e+01
<code>%s</code>	a string	"Java is cool"

```
int count = 5;
double amount = 45.56;
System.out.printf("count is %d and amount is %f", count, amount);
```

display count is 5 and amount is 45.560000

FormatDemo

The example gives a program that uses `printf` to display a table.

[FormatDemo](#)

Text Blocks

Text blocks became a standard feature since Java 15. It enables you to form a multi-line string. Here is an example of a text block:

```
String tb = """
    Java
    Python
    C++""";
System.out.println(tb);
```

The code displays three lines

```
Java
Python
C++
```

A text block contains multiple line. The first line starts with three double quotation marks (""") followed by space characters. The last line ends with three double quotation marks (""").

Indentation in Text Blocks

You can use indentation in a text block. To achieve this, the compiler first removes the maximum left common space characters for all lines. For example, the maximum common left space characters for all lines in the following text block is 2, because there are two leading space characters before "Java".

```
String tb = ""  
  Java  
    Python  
      C++"";
```

After removing two leading space characters from each line, the resulting string is

```
Java  
  Python  
    C++
```

After removing the maximum common left space characters, the indentation of the text block is preserved.

Indentation in Text Blocks: ending """

Note if the ending """ is on a separate line by itself, its leading spaces are counted in the maximum common left space. For example, in the following code,

```
String tb = """  
    Java  
        Python  
            C++  
""";
```

The maximum common left space is 0, because there are no space before the ending """. The resulting string will be

```
Java  
    Python  
        C++
```

Right Trailing Space

The right trailing space are trimmed by default. If you want to keep right trailing space, use `\s`. For example, the following code keep three right space characters after "Python".

```
String tb = """  
    Java  
        Python \s  
            C++""";
```

Note that there are two spaces after `Python` and before `\s`. Including `\s`, there will be three spaces after "Python".

Escape Characters

You used `\s` escape character for a space. You can also use other escape characters in needed. For example, for the following text block:

```
String tb = """
Java
\tPython\n
C++""";
```

The resulting string is

```
Java
    Python
C++
```

The space before `"Python"` is due to the tab character (`\t`).
The space line after `"Python"` is due to the new line character `"\n"`.

""" in Text Block as Literal

If you need to put """ into the text block, you can use \ """. For example, for the following text block:

```
String tb = """  
    Java\ "" "  
    Python  
    C++ "" " ;
```

The resulting string is

```
Java "" "  
Python  
C++
```

New String Methods

As part of the new feature on text blocks, several new String methods are introduced. You can use the `stripIndent()` to remove the maximum common left space from a multi-line string in the same way as a text block is processed.

You can use the `formatted(args)` method to format a string. For example, for the following code:

```
String tb = """
    Product: %s
    Price: $%.2f""".formatted("Salt", 4.52);
```

The resulting string is

```
Produce: Salt
Price: $4.52
```

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.