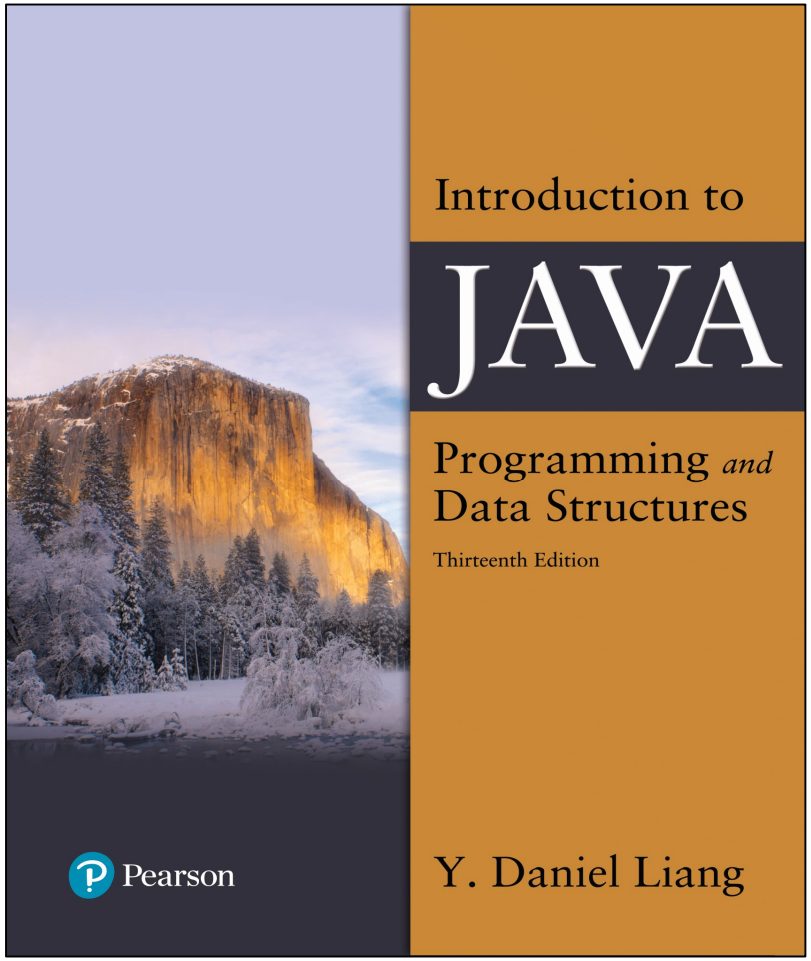
Introduction to Java Programming and Data Structures

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

Exponent Methods

- **`pow(double a, double b)`**
  Returns `a` raised to the power of `b`.
- **`sqrt(double a)`**
  Returns the square root of `a`.

Examples:

```
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).

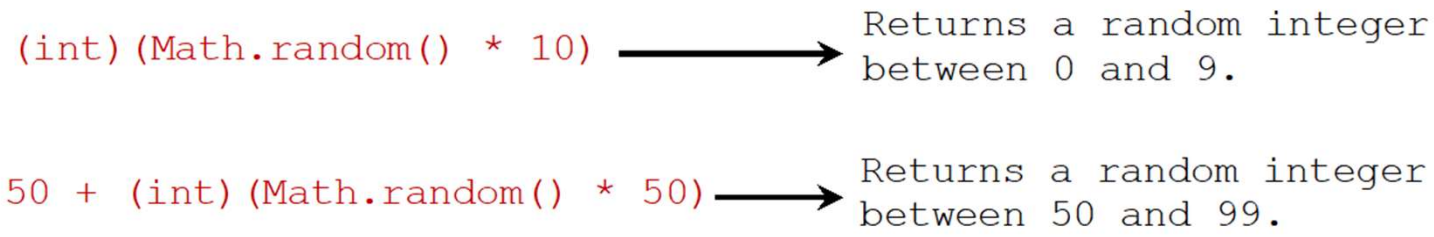
Examples:

```
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
```

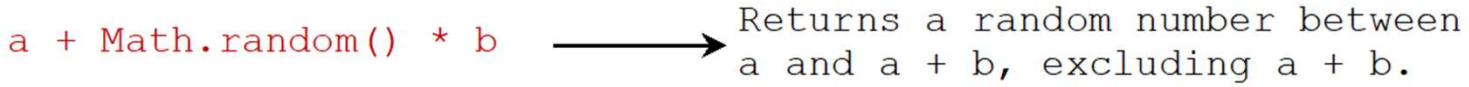
The `random` Method

Generates a random **`double`** value greater than or equal to 0.0 and less than 1.0 (**`0 <= Math.random() < 1.0`**).

Examples:

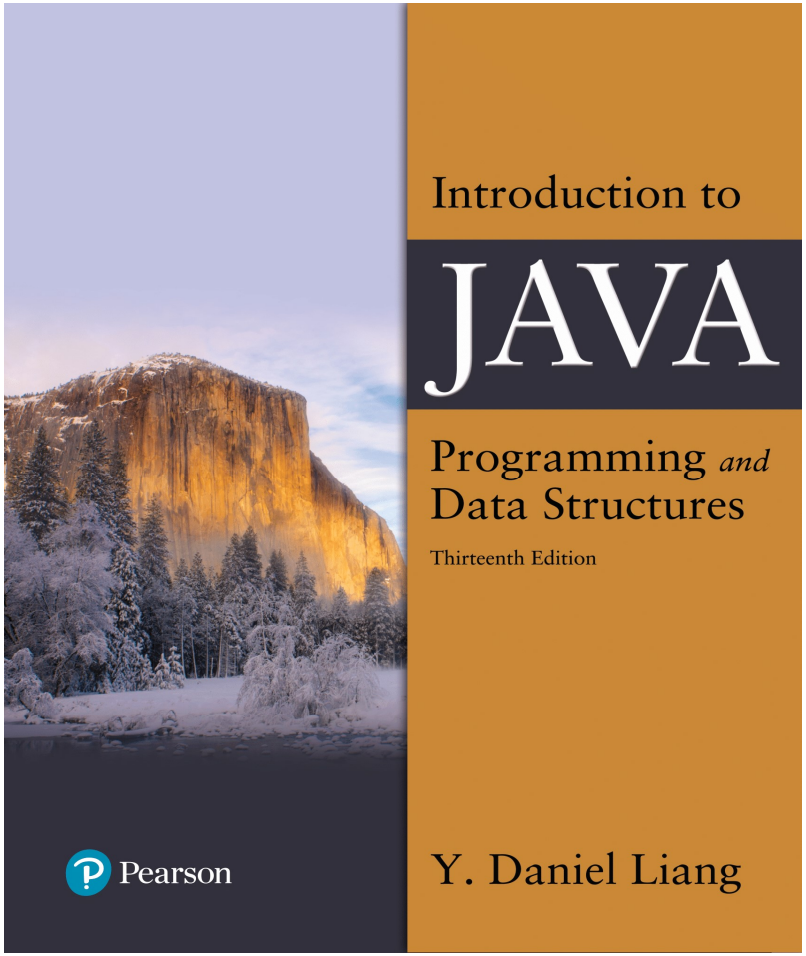


In general,



Introduction to Java Programming and Data Structures

Thirteenth Edition



Appendix F

Number Systems

Number Systems (1 of 4)

binary	0, 1
octal	0, 1, 2, 3, 4, 5, 6, 7
decimal	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
hexdecimal	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Number Systems (2 of 4)

Computers use binary numbers internally because storage devices like memory and disk are made to store 0s and 1s. A number or a text inside a computer is stored as a sequence of 0s and 1s. Each 0 and 1 is called a **bit**, short for **bi**nary digi**t**. The binary number system has two digits, 0 and 1.

Binary numbers are not intuitive, since we use decimal numbers in our daily life. When you write a number like 20 in a program, it is assumed to be a decimal number. Internally, computer software is used to convert decimal numbers into binary numbers, and vice versa.

Number Systems (3 of 4)

The digits in the decimal number system are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. A decimal number is represented using a sequence of one or more of these digits. The value that each digit in the sequence represents depends on its position. A position in a sequence has a value that is an integral power of 10. For example, the digits 7, 4, 2, and 3 in decimal number 7423 represent 7000, 400, 20, and 3, respectively, as shown below:

7	4	2	3
10^3	10^2	10^1	10^0

$$= 7 \times 10^3 + 4 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$$

$$= 7000 + 400 + 20 + 3 \quad = 7423$$

The decimal number system has ten digits and the position values are integral powers of 10. We say that 10 is the **base** or **radix** of the decimal number system. Similarly, the base of the binary number system is 2 since the binary number system has two digits and the base of the hex number system is 16 since the hex number system has sixteen digits.

Number Systems (4 of 4)

- Binary numbers tend to be very long and cumbersome. Hexadecimal numbers are often used to abbreviate binary numbers.
- The octal number system has 8 digits: 0, 1, 2, 3, 4, 5, 6, 7
- The hexadecimal number system has 16 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The letters A, B, C, D, E, and F correspond to the decimal numbers 10, 11, 12, 13, 14, and 15.

Binary Numbers => Decimals

Given a binary number $b_n b_{n-1} b_{n-2} \ldots b_2 b_1 b_0$ the equivalent decimal value is

$$b_n \times 2^n + b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \ldots + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

`10` in binary $1 \times 2^1 + 0$ = `2` in decimal

`1000` in binary $1 \times 2^3 + 0 \times 2^2 + 0 \times 2 + 0$ = `8` in decimal

`10101011` in binary $1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 1$

= `171` in decimal

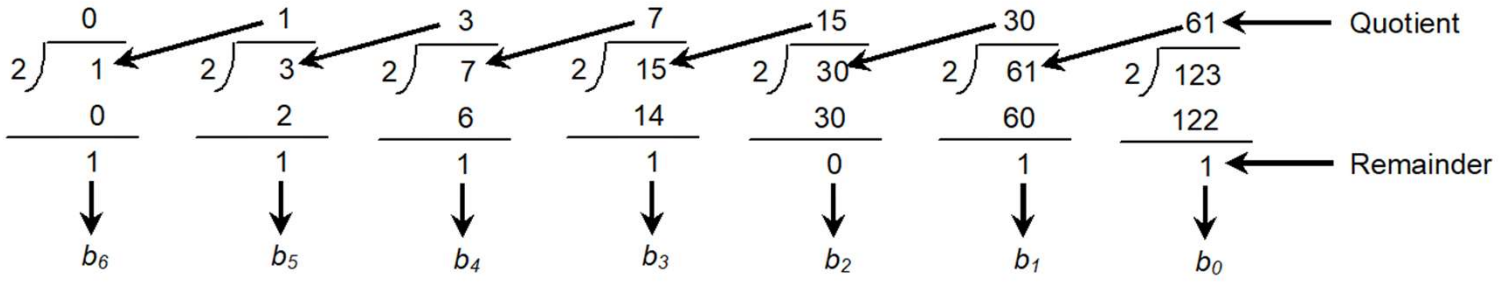
Decimals => Binary

To convert a decimal number d to a binary number is to find the binary digits.. $b_n, b_{n-1}, b_{n-2}, \ldots, b_2, b_1, b_0$ such that

$$d = b_n \times 2^n + b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \ldots + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

These numbers can be found by successively dividing d by 2 until the quotient is 0. The remainders are $b_0, b_1, b_2, \ldots, b_{n-2}, b_{n-1}, b_n$

For example, the decimal number 123 is 1111011 in binary. The conversion is conducted as follows:



Octal Numbers => Decimals

Given a binary number $O n O n-1 O n-2 \ldots O 2 O 1 O 0$ the equivalent decimal value is

$$O n \times 8^{n} + O n-1 \times 8^{n-1} + O n-2 \times 8^{n-2} + \ldots + O 2 \times 8 + O 1 \times 8 + O 0 \times 8^{0}$$

`10` in Octal $1 \times 8^{1} + 0 \times 8^{0}$ = `8` in decimal

`2070` in binary $2 \times 8^{3} + 0 \times 8^{2} + 7 \times 8^{1} + 0 \times 8^{0}$ = `1080` in decimal

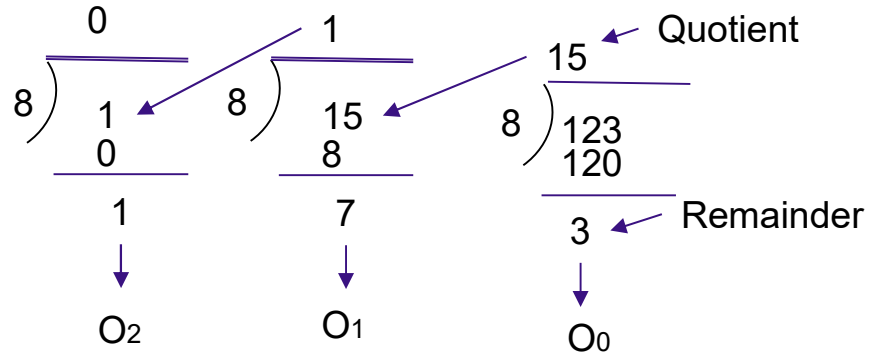
Decimals => Octal

To convert a decimal number d to a binary number is to find the binary digits.. $On, On-1, On-2, \ldots, O2, O1, O0$ such that

$$d = On \times 8^n + On-1 \times 8^{n-1} + On-2 \times 8^{n-2} + \ldots + O2 \times 8^2 + O1 \times 8^1 + O0 \times 8^0$$

These numbers can be found by successively dividing d by 2 until the quotient is 0. The remainders are $O0, O1, O2, \ldots, On-2, On-1, On$

For example, the decimal number 123 is 173 in octal. The conversion is conducted as follows:



Hexadecimals to Decimals

The hexadecimal number system has sixteen digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. The letters A, B, C, D, E, and F correspond to the decimal numbers 10, 11, 12, 13, 14, and 15. Given a hexadecimal number

$h_n h_{n-1} h_{n-2} \ldots h_2 h_1 h_0$

The equivalent decimal value is

$$h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \ldots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

`7F` in hex $7 \times 16^1 + 15$ = `127` in decimal

`FFFF` in hex $15 \times 16^3 + 15 \times 16^2 + 15 \times 16 + 15$

= `65535` in decimal

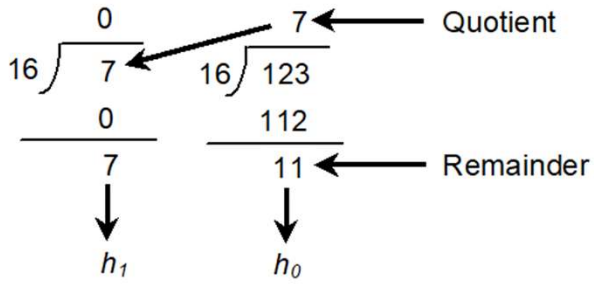
Decimals => Hexadecimal

To convert a decimal number d to a hexadecimal number is to find the hexadecimal digits … $h_n, h_{n-1}, h_{n-2}, \ldots, h_2, h_1, h_0$ such that

$$d = h_n \times 16^n + h_{n-1} \times 16^{n-1} + h_{n-2} \times 16^{n-2} + \ldots + h_2 \times 16^2 + h_1 \times 16^1 + h_0 \times 16^0$$

These numbers can be found by successively dividing d by 16 until the quotient is 0. The remainders are $h_0, h_1, h_2, \ldots, h_{n-2}, h_{n-1}, h_n$

For example, the decimal number 123 is 7B in hexadecimal. The conversion is conducted as follows:

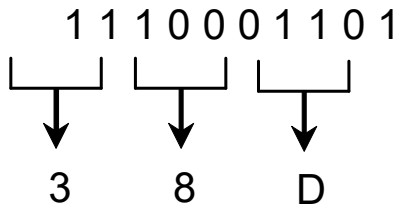


Hexadecimal ⇔ Binary

Decimal	Binay	Octal	Headecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

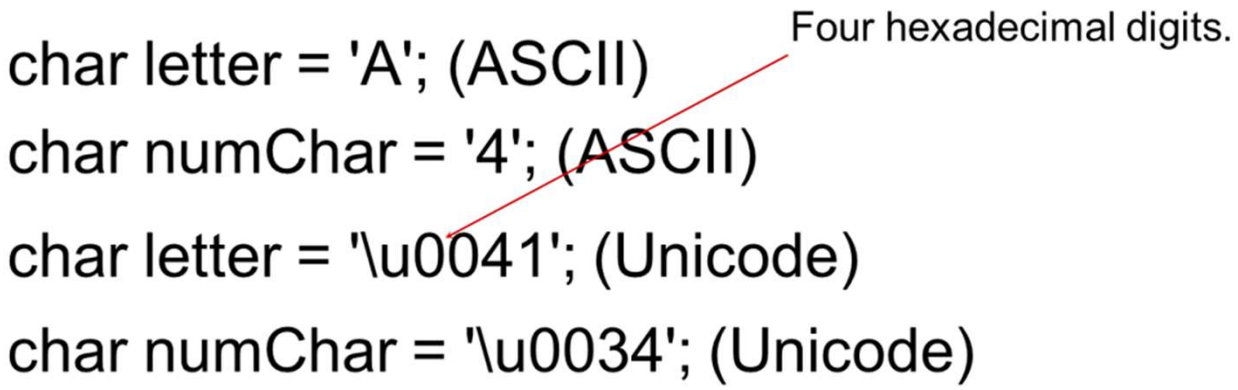
To convert a hexadecimal number to a binary number, simply convert each digit in the hexadecimal number into a four-digit binary number.

To convert a binary number to a hexadecimal, convert every four binary digits from right to left in the binary number into a hexadecimal number. For example,



Pearson

`Character` Data Type

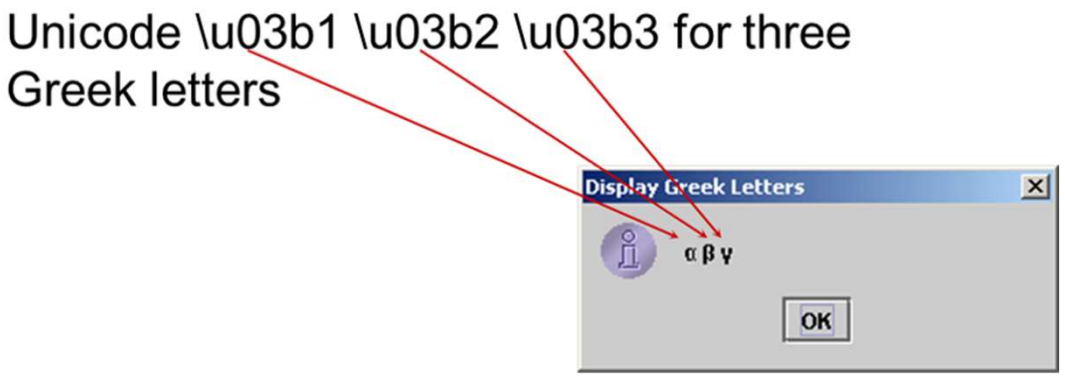


Note: The increment and decrement operators can also be used on **`char`** variables to get the next or preceding Unicode character. For example, the following statements display character **b**.

```
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`.



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
\b	Backspace	**\u0008**	8
\t	Tab	**\u0009**	9
\n	Linefeed	**\u000A**	10
\f	Formfeed	**\u000C**	12
\r	Carriage Return	**\u000D**	13
****	Backslash	**\u005C**	92
\”	Double Quote	**\u0022**	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		

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
`isDigit(ch)`	Returns true if the specified character is a digit.
`isLetter(ch)`	Returns true if the specified character is a letter.
`isLetterOfDigit(ch)`	Returns true if the specified character is a letter or digit.
`isLowerCase(ch)`	Returns true if the specified character is a lowercase letter.
`isUpperCase(ch)`	Returns true if the specified character is an uppercase letter.
`toLowerCase(ch)`	Returns the lowercase of the specified character.
`toUpperCase(ch)`	Returns the uppercase of the specified character.

Formatting Output

Use the `printf` statement.

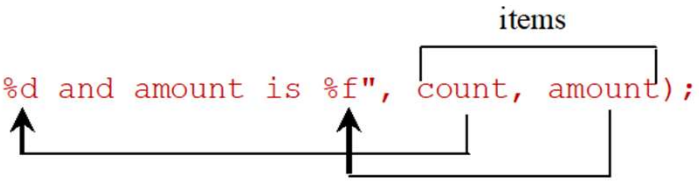
```
System.out.printf(format, items);
```

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
%b	a boolean value	true or false
%c	a character	'a'
%d	a decimal integer	200
%f	a floating-point number	45.460000
%e	a number in standard scientific notation	4.556000e+01
%s	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

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

Simple Methods for `String` Objects (2 of 2)

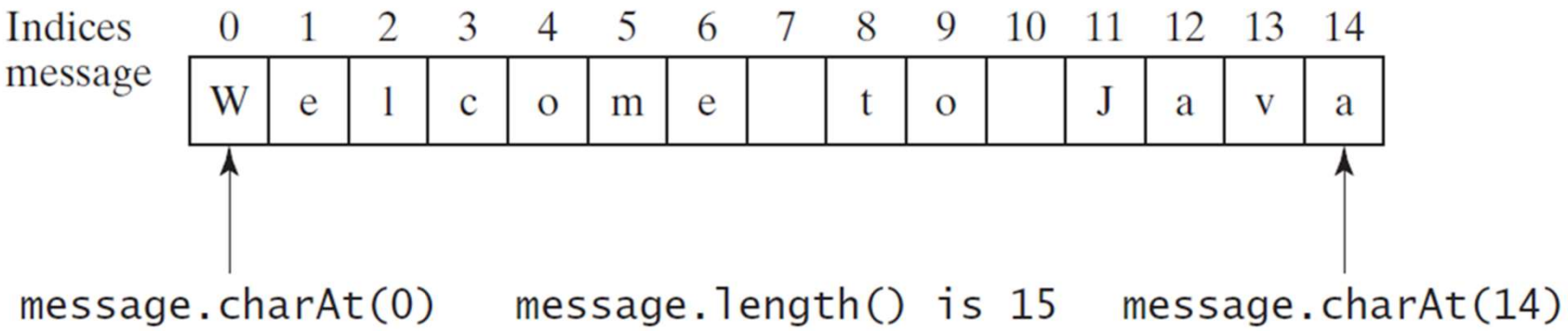
Strings are objects in Java. The methods in the preceding table can only be invoked from a specific string instance. For this reason, these methods are called **instance methods**. A non-instance method is called a **static method**. A static method can be invoked without using an object. All the methods 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 to java ".trim() returns a new string, Welcome to java.

`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);
```

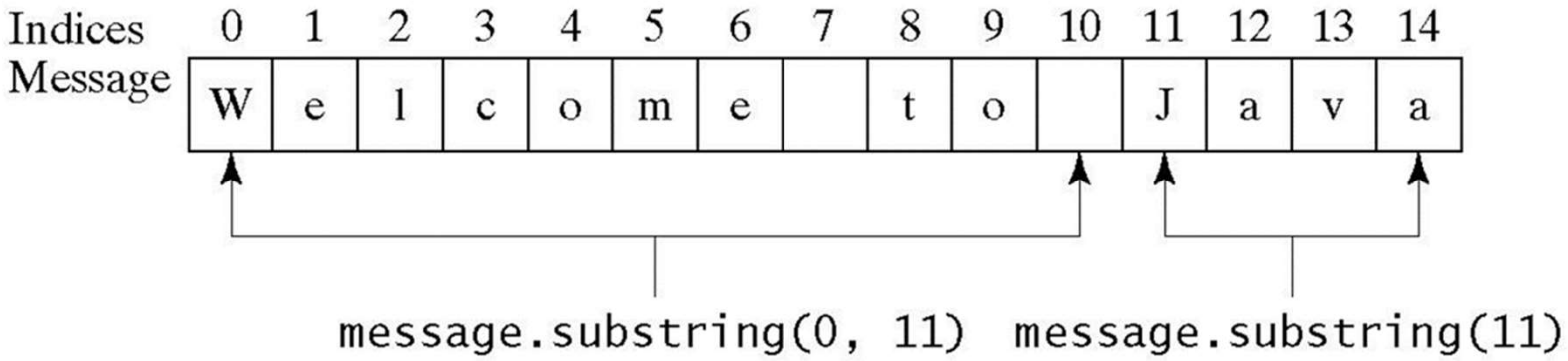
Comparing Strings

Method	Description
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	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 `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.

OrderTwoCities

Obtaining Substrings

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

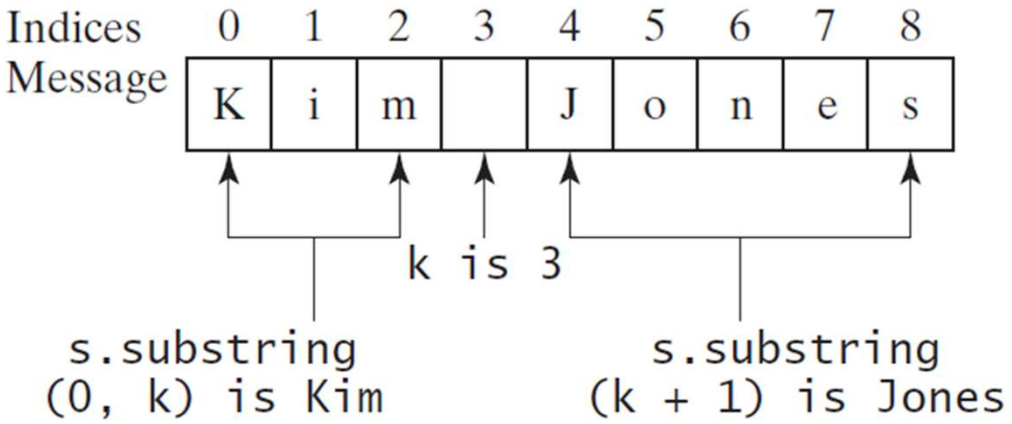


Finding a `Character` or a `Substring` in a `String` (1 of 2)

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