# Chapter 4 Mathematical Functions, Characters, and Strings

胡 铮
huzheng@bupt.edu.cn

## 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)`

`a: Radians` 弧度

- `acos(double b)`
- `asin(double b)`
- `atan(double b)`

  - `toRadians(90)`
  - `toDegrees(Math.PI/2`
    `)`

```
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

```
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
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:

```
(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.

## Examples

- ✦ MathTest.java
- ✦ ComputeAngles.java

# Character Data Type

char letter = 'A'; (ASCII)    // *literals*

char numChar = '4'; (ASCII)

Four hexadecimal digits.

char letter = '\u0041'; (Unicode)

char numChar = '\u0034'; (Unicode)

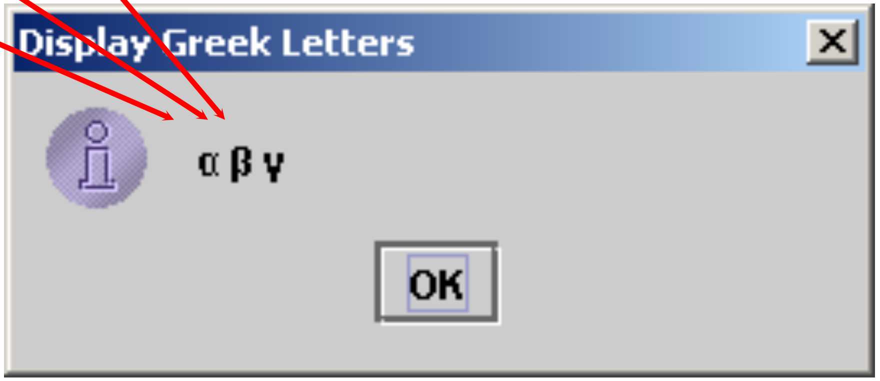
NOTE: The increment and decrement operators can also be used on char variables to get the next or preceding Unicode character.

```
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* |
|---|---|---|---|
| `\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

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 | | |

# ASCII Character Set, cont.

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

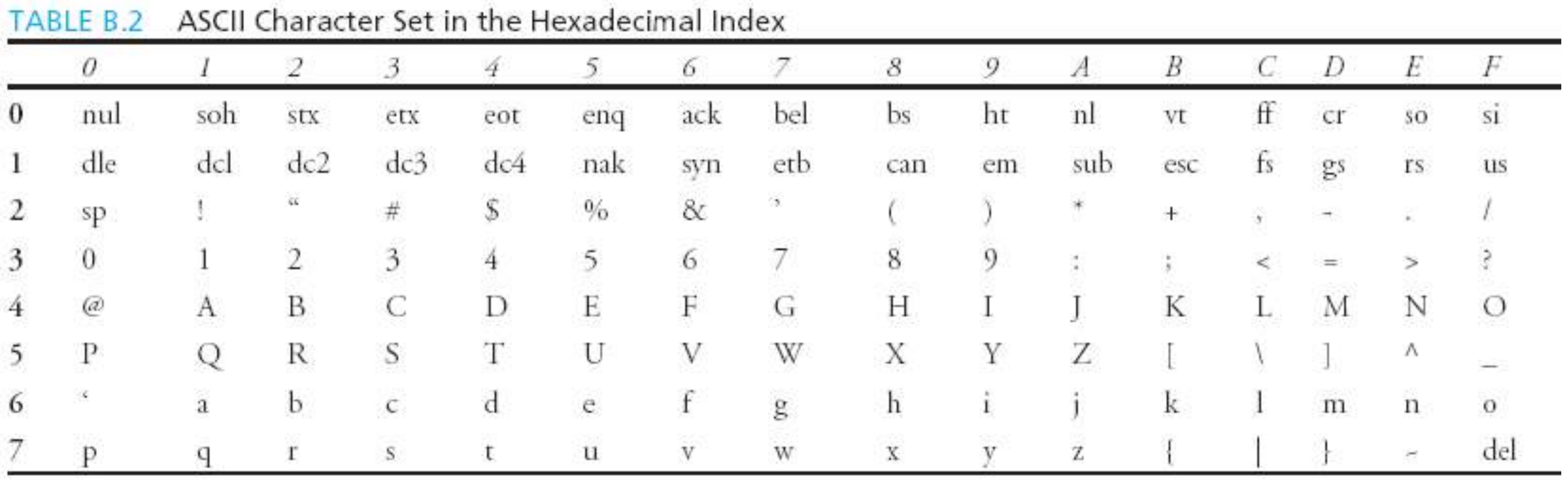


TABLE B.2 ASCII Character Set in the Hexadecimal Index

| | *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** |
|---|---|
| `isDigit(ch)` | Returns true if the specified character is a digit. |
| `isLetter(ch)` | Returns true if the specified character is a letter. |
| `isLetterOrDigit(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. |

# The String Type

To represent a string of characters, use the data type called String (*reference type*, not primitive type)

```
String message = "Welcome to Java";
String message = new String(“Welcome to
Java”);
```

String is actually a predefined class like the System class and Scanner class.

Any Java class can be used as a reference type for a variable.

# Simple Methods for **String** Objects

| **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

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)**.

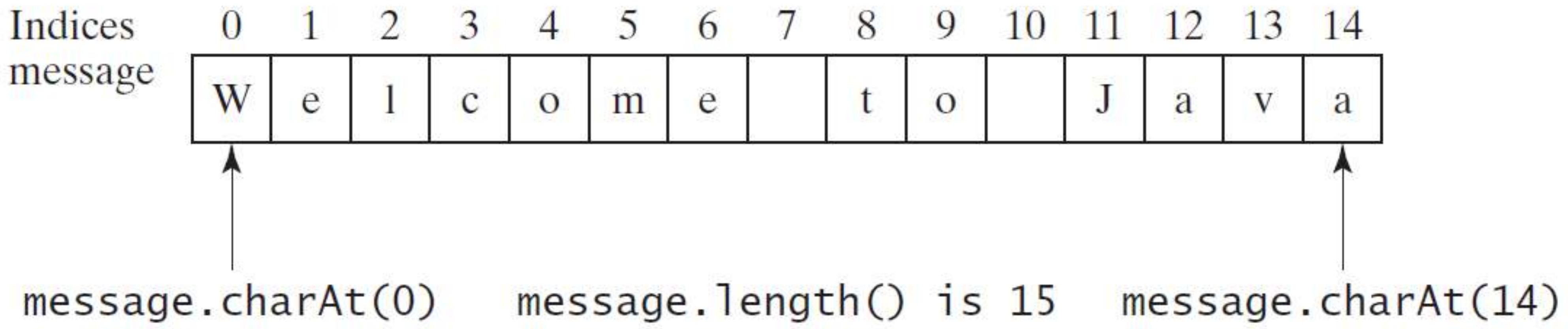
## Getting String Length

```
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);
```

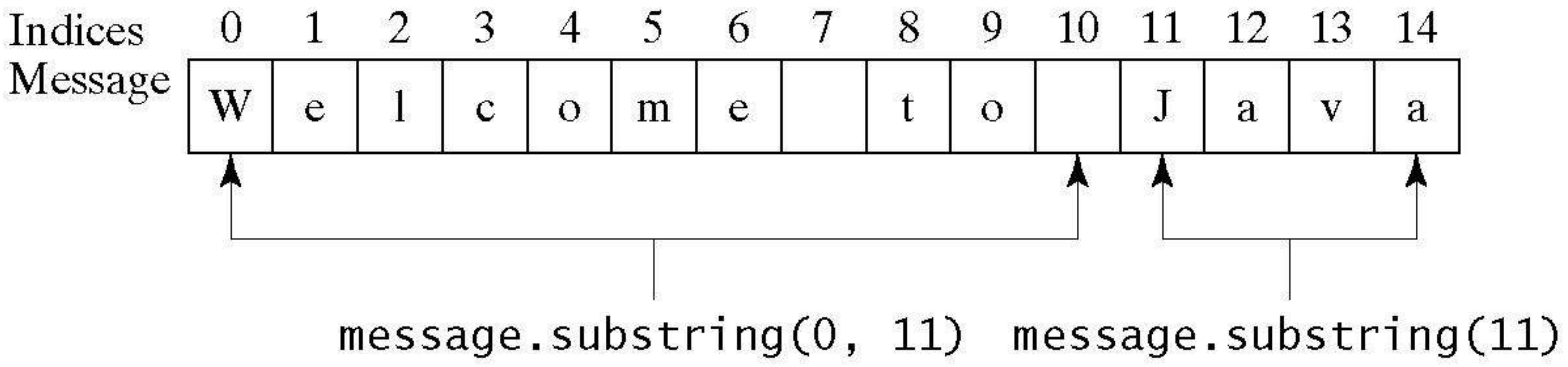
## Comparing Strings

| **Method** | **Description** |
|---|---|
| `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. |

ch04/OrderTwoCities.java

# 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. |

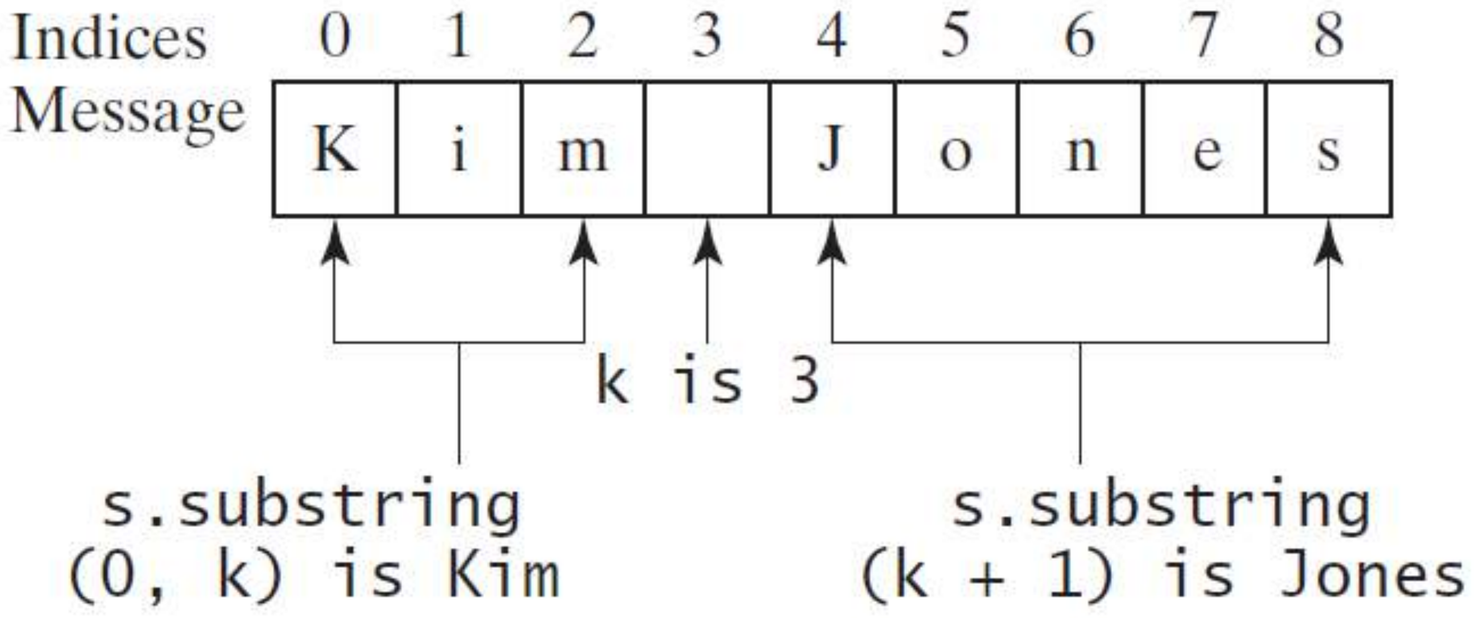


# Finding a Character or a Substring in a String

| 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

```
int k = s.indexOf(' ');
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```



Indices 0 1 2 3 4 5 6 7 8
Message K i m J o n e s
k is 3
s.substring (0, k) is Kim
s.substring (k + 1) is Jones

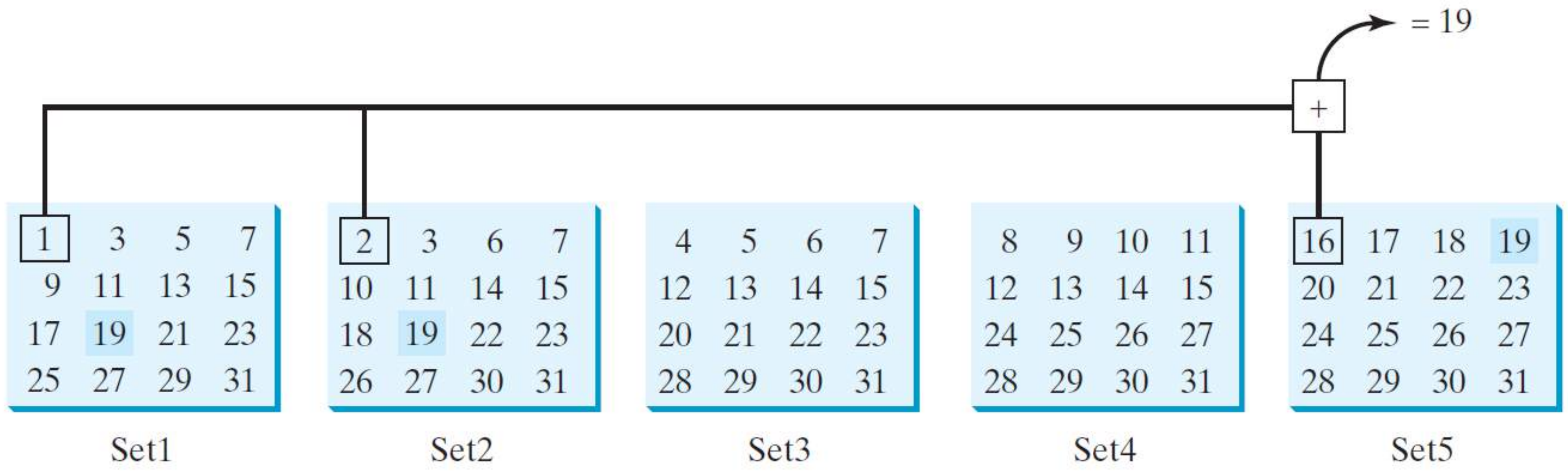
# Conversion between Strings and Numbers

```
int intValue = Integer.parseInt(intString);
double doubleValue =
Double.parseDouble(doubleString);

String s = number + "";
```

## Problem: Guessing Birthday

The program can guess your birth date. Run to see how it works.



Ch04/GuessBirthday.java

# Mathematics Basis for the Game

19 is 10011 in binary. 7 is 111 in binary. 23 is 11101 in binary

```
                                         10000
   10000          00110                   1000
      10             10                    100
+      1       +      1               +      1
--------       --------               --------
   10011          00111                  11101

    19              7                      23
```

| Decimal | Binary |
|---|---|
| 1 | 00001 |
| 2 | 00010 |
| 3 | 00011 |
| ... | |
| 19 | 10011 |
| ... | |
| 31 | 11111 |

$$
\begin{array}{r}
b_5\;0\;0\;0\;0 \\
b_4\;0\;0\;0 \\
b_3\;0\;0 \\
b_2\;0 \\
+\qquad\qquad b_1 \\
\hline
b_5\,b_4\,b_3\,b_2\,b_1
\end{array}
\qquad
\begin{array}{r}
10000 \\
10 \\
+\qquad 1 \\
\hline
10011 \\
19
\end{array}
\qquad
\begin{array}{r}
10000 \\
1000 \\
100 \\
10 \\
+\qquad 1 \\
\hline
11111 \\
31
\end{array}
$$

## Case Study: Converting a Hexadecimal Digit to a Decimal Value

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

ch04/HexDigit2Dec.java

## Case Study: Revisting the Lottery Program Using Strings

A problem can be solved using many different approaches. This section rewrites the lottery program in ch03/Lottery.java using strings. Using strings simplifies this program.

ch04/LotteryUsingStrings.java

## 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);
```

items

```
display             count is 5 and amount is 45.560000
```

# FormatDemo

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

ch04/FormatDemo.java

## Scanner for String

next() vs. nextLine()

- using nextLine() and then parse the string.
- Using next() will only return what comes before a space. nextLine() automatically moves the scanner down after returning the current line.

ch04/ScannerDemo.java