

COMP30820  
Java Programming (Conv)

Michael O'Mahony

# Chapter 4 Mathematical Functions, Characters, and Strings

# Objectives

- ◆ To solve mathematics problems by using the methods in the `Math` class (§4.2).
- ◆ To represent characters using the `char` type (§4.3).
- ◆ To encode characters using ASCII and Unicode (§4.3.1).
- ◆ To represent special characters using the escape sequences (§4.4.2).
- ◆ To cast a numeric value to a character and cast a character to an integer (§4.3.3).
- ◆ To compare and test characters using the static methods in the `Character` class (§4.3.4).
- ◆ To represent strings using the `String` objects (§4.4).
- ◆ To return the string length using the `length()` method (§4.4.1).
- ◆ To return a character in the string using the `charAt(i)` method (§4.4.2).
- ◆ To use the `+` operator to concatenate strings (§4.4.3).
- ◆ To read strings from the console (§4.4.4).
- ◆ To read a character from the console (§4.4.5).
- ◆ To compare strings using the `equals` method and the `compareTo` methods (§4.4.6).
- ◆ To obtain substrings (§4.4.7).
- ◆ To find a character or a substring in a string using the `indexOf` method (§4.4.8).
- ◆ To format output using the `System.out.printf` method (§4.6).

# The Math Class

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

Class constants:

- `PI` (the ratio of the circumference of a circle to its diameter)
- `E` (base of natural logarithms)

Class methods:

- Exponent Methods
- Rounding Methods
- `random` Method

See the Java API for more information –

<https://docs.oracle.com/en/java/javase/11/docs/api/index.html>

# Math Methods

- ✦ **double exp(double a)**

Returns e raised to the power of a

Example: `double x = Math.exp(1); // returns Math.E (~2.71)`

- ✦ **double log(double a)**

Returns the natural logarithm (base e) of a

Example: `double x = Math.log(Math.E); // returns 1.0`

- ✦ **double log10(double a)**

Returns the base 10 logarithm of a

Example: `double x = Math.log10(100); // returns 2.0`

- ✦ **Note: log(double a) / log(double B)**

Returns the base B logarithm of a

Example: `double x = Math.log(256) / Math.log(2); // i.e.  $\log_2(256)$   
// returns 8`

# Math Methods

## ✦ **double pow(double a, double b)**

Returns a raised to the power of b

Example: `double x = Math.pow(2, 3); // returns 8.0`

## ✦ **double sqrt(double a)**

Returns the square root of a

Example: `double x = Math.sqrt(4); // returns 2.0`

# Rounding Methods

- ✦ **double ceil(double a)**

a is rounded up to its nearest integer

Example: `double x = Math.ceil(10.1); // returns 11.0`

- ✦ **double floor(double a)**

a is rounded down to its nearest integer

Example: `double x = Math.floor(10.9); // returns 10.0`

- ✦ **long round(double a)**

Returns the closest long to a

Example: `long x = Math.round(9.5); // returns 10`

Example: `long x = Math.round(10.5); // returns 11`

- ✦ **Rounding a number to n decimal places – examples:**

`double x = 2.0 / 3; // x is 0.6666666666666666`

`double r2 = Math.round(x * 100) / 100.0; // r2 is 0.67`

`double r3 = Math.round(x * 1000) / 1000.0; // r3 is 0.667`

# random Method

- ✦ **double random()** – returns a random `double` value in the interval  $[0, 1)$



# random Method

- ✦ **double random()** – returns a random double value in the interval [0, 1)

## Examples:

Generate a random double in the interval [0, 1)

```
double x = Math.random();
```

# random Method

- ✦ **double random()** – returns a random double value in the interval [0, 1)

## Examples:

Generate a random double in the interval [0, 1)

```
double x = Math.random();
```

Generate a random double in the interval [0, 10)

```
double x = Math.random() * 10;
```

# random Method

- ✦ **double random()** – returns a random `double` value in the interval `[0, 1)`

## Examples:

Generate a random `double` in the interval `[0, 1)`

```
double x = Math.random();
```

Generate a random `double` in the interval `[0, 10)`

```
double x = Math.random() * 10;
```

Generate a random `int` in the interval `[0, 9]` (between 0 and 9, inclusive)

```
int x = (int)(Math.random() * 10);
```

# random Method

- ♦ **double random()** – returns a random `double` value in the interval `[0, 1)`

## Examples:

Generate a random `double` in the interval `[0, 1)`

```
double x = Math.random();
```

Generate a random `double` in the interval `[0, 10)`

```
double x = Math.random() * 10;
```

Generate a random `int` in the interval `[0, 9]` (between 0 and 9, inclusive)

```
int x = (int)(Math.random() * 10);
```

Generate a random `int` in the interval `[50, 99]` (between 50 and 99, inclusive)

```
int x = 50 + (int)(Math.random() * 50)
```

# random Method

- ♦ **double random()** – returns a random double value in the interval [0, 1)

## Examples:

Generate a random double in the interval [0, 1)

```
double x = Math.random();
```

Generate a random double in the interval [0, 10)

```
double x = Math.random() * 10;
```

Generate a random int in the interval [0, 9] (between 0 and 9, inclusive)

```
int x = (int)(Math.random() * 10);
```

Generate a random int in the interval [50, 99] (between 50 and 99, inclusive)

```
int x = 50 + (int)(Math.random() * 50)
```

In general, the following return a random number in the interval [a, a+b)

```
int x = a + (int)(Math.random() * b);
```

```
double x = a + Math.random() * b;
```

# Characters

Computers use binary numbers internally. A character is stored in a computer as a sequence of 0s and 1s.

Mapping a character to its binary representation is called *encoding*. There are different ways to encode a character – defined by an *encoding scheme*.

Common encoding schemes:

- ASCII (American Standard Code for Information Interchange): 8-bit encoding scheme (128 characters). Represents all uppercase and lowercase letters, digits, punctuation marks, and control characters.
- 16-bit Unicode characters (65,536 characters) can be stored in a `char` type variable.

Note: the Unicode standard has been extended to allow additional characters:

- Those characters that go beyond the original 16-bit limit are called *supplementary characters*.
- While Java supports supplementary characters, we do not consider these here.

# char Data Type

The character data type `char` represents a single character.

A character literal is enclosed in single quotation marks:

```
char ch = 'A';  
char n = '4';
```

Note:

- A string literal is enclosed in quotation marks (" ").
- For example, "A" is a string, but 'A' is a character.

# char Data Type

Unicode takes two bytes (16 bits), preceded by `\u`, expressed in four hexadecimal digits that run from `'\u0000'` to `'\uFFFF'`

Unicode includes ASCII code, with `'\u0000'` to `'\u007F'` corresponding to the 128 ASCII characters.

```
char letter = 'A'; // (ASCII)
char ch = '4'; // (ASCII)
```

...is equivalent to...

```
char letter = '\u0041'; // (Unicode)
char ch = '\u0034'; // (Unicode)
```



# ASCII Code & Unicode 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

# Casting between `char` and Numeric Types

A `char` can be cast into any numeric type, and vice versa.

When an integer is cast into a `char`, only its lower 16 bits of data are used; the other part is ignored (since Unicode takes 16 bits). For example:

```
char c = (char)65; // ch is 'A'
```

When a floating-point value is cast into a `char`, the floating-point value is first cast into an `int`, which is then cast into a `char`:

```
char ch = (char)65.25; // 65 is assigned to ch; ch is 'A'
```

When a `char` is cast into a numeric type, the character's Unicode is cast into the specified numeric type.

```
int i = (int)'A'; // The Unicode of 'A' is assigned to i;  
                // i is 65
```

# Casting between `char` and Numeric Types

All numeric operators can be applied to `char` operands. A `char` operand is automatically cast into a number if the other operand is a number or a character.

If the other operand is a string, the character is concatenated with the string.

For example:

```
int i = '2' + '3'; // (int)'2' is 50 and (int)'3' is 51; i is 101
int j = 2 + 'a'; // (int)'a' is 97; j is 99
char ch = 2 + 'a'; // (int)'a' is 97; ch is 'c'
System.out.println("Chapter " + '4'); // displays "Chapter 4"
```

# Character Data Type

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

For example:

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

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

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

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

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

# Character Data Type

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

For example:

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

```
System.out.println(ch++); // prints 'a'
```

```
System.out.println(ch);   // prints 'b'
```

```
System.out.println(--ch);  // prints 'a'
```

```
System.out.println(ch);   // prints 'a'
```

# Escape Sequences for Special Characters

<i>Escape Sequence</i>	<i>Name</i>	<i>Unicode Code</i>	<i>Decimal Value</i>
<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

## Examples:

```
char ch = '\\'; // ch is \
```

```
char ch = '\"'; // ch is "
```

# Comparing and Testing Characters

Two characters can be compared using the relational operators, just like comparing two numbers.

This is done by comparing the Unicodes of the two characters; for example:

'a' < 'b' is true because the Unicode for 'a' (97) is less than the Unicode for 'b' (98).

'a' < 'A' is false because the Unicode for 'a' (97) is greater than the Unicode for 'A' (65).

'1' < '8' is true because the Unicode for '1' (49) is less than the Unicode for '8' (56).

# Comparing and Testing Characters

```
char ch = 'x';

// check if ch is an uppercase letter
if (ch >= 'A' && ch <= 'Z')
    System.out.println(ch + " is an uppercase letter");

// check if ch is a lowercase letter
if (ch >= 'a' && ch <= 'z')
    System.out.println(ch + " is a lowercase letter");

// check if ch is a digit
if (ch >= '0' && ch <= '9')
    System.out.println(ch + " is a digit");
```



# Methods in the Character Class

For convenience, Java provides methods in the `Character` class for testing and converting characters

<i>Method</i>	<i>Description</i>
<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.

# Comparing and Testing Characters

```
char ch = 'x';

// check if ch is an uppercase letter
if (Character.isUpperCase(ch))
    System.out.println(ch + " is an uppercase letter");

// check if ch is a lowercase letter
if (Character.isLowerCase(ch))
    System.out.println(ch + " is a lowercase letter");

// check if ch is a digit
if (Character.isDigit(ch))
    System.out.println(ch + " is a digit");
```

# Converting Characters

```
// Convert to uppercase  
char c1 = 'a';  
char c2 = Character.toUpperCase(c1); // c2 is 'A'
```

```
// Convert to lowercase  
char c3 = 'A';  
char c4 = Character.toLowerCase(c3); // c4 is 'a'
```

# The String Type

- ◆ The `char` type represents a single character.
- ◆ To represent a string of characters, use the `String` data type. For example,  

```
String message = "Welcome to Java";
```
- ◆ `String` is a predefined class in the Java API (like the `Scanner` class).
- ◆ Note: the `String` type is not a primitive type – it is known as a *reference type*:
  - In the above, `message` is a reference variable that references a string object with contents `"Welcome to Java"`
  - Reference data types will be covered in Chapter 9 (Objects and Classes).
- ◆ For now, you just need to know how to declare a `String` variable, how to assign a string to the variable, and to perform operations for strings.

# Some Methods for `String` Objects

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.

# Getting String Length and Characters from a String

Example:

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

Indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	W	e	l	c	o	m	e		t	o		J	a	v	a

```
int len = message.length();    // len is 15
```

```
char c1 = message.charAt(0);    // c1 is 'W'
```

```
char c2 = message.charAt(14);   // c2 is 'a'
```

# String Concatenation

## Example:

```
String s1 = "Hello ";  
String s2 = "World";  
  
String s3 = s1.concat(s2); // s3 is "Hello World"
```

...or...

```
String s3 = s1 + s2; // s3 is "Hello World"
```

## Further examples:

```
// three strings are concatenated  
String s1 = "Welcome " + "to " + "Java";  
  
// string "Chapter" is concatenated with number 2  
String s2 = "Chapter " + 2; // s2 is "Chapter 2"  
  
// string "Supplement" is concatenated with character 'B'  
String s3 = "Supplement " + 'B'; // s3 is "Supplement B"
```

# Converting Strings

```
String s1 = "Welcome";
```

```
String s2 = s1.toLowerCase(); // returns a new string, "welcome"
```

```
String s3 = s1.toUpperCase(); // returns a new string, "WELCOME"
```

```
String s1 = "    Hi there!    ";
```

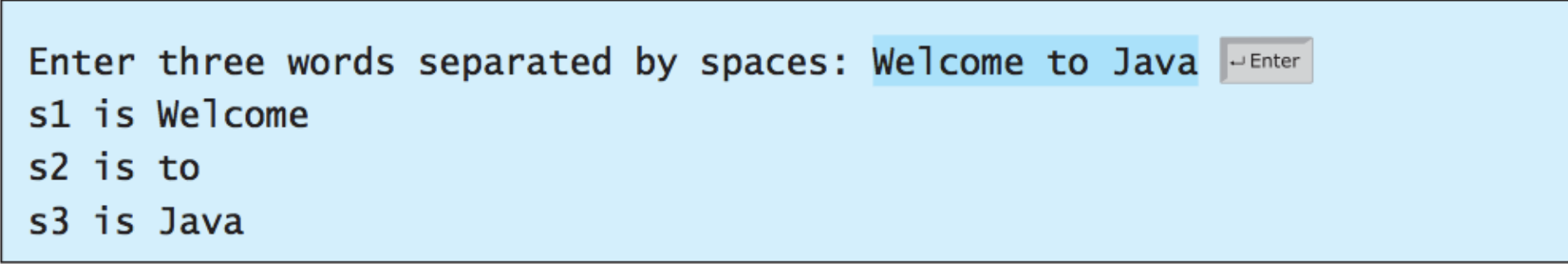
```
String s2 = s1.trim(); // returns a new string, "Hi there!"
```



# Reading a String from the Console

To read a string from the console, invoke the `next()` method on a `Scanner` object. For example, the following code reads three strings from the keyboard:

```
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);
```

A screenshot of a Java application window with a light blue background. It shows the execution of the provided code. The prompt "Enter three words separated by spaces:" is followed by the user input "Welcome to Java" which is highlighted in blue. To the right of the input is a grey button with a right arrow and the text "Enter". Below the input, the program's output is displayed: "s1 is Welcome", "s2 is to", and "s3 is Java".

```
Enter three words separated by spaces: Welcome to Java Enter
s1 is Welcome
s2 is to
s3 is Java
```


# Reading a String from the Console

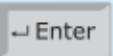
The `next()` method reads a string that ends with a whitespace character.

Use the `nextLine()` method to read an entire line of text. The `nextLine()` method reads a string that ends with the *Enter* key pressed.

For example, the following statements read a line of text:

```
Scanner input = new Scanner(System.in);  
System.out.println("Enter a line: ");  
String s = input.nextLine();  
System.out.println("The line entered is " + s);
```



```
Enter a line: Welcome to Java   
The line entered is Welcome to Java
```

# Reading a Character from the Console

To read a character from the console, use the `next()` method to read a string, then invoke the `charAt(index)` method on the string to return a character:

```
Scanner input = new Scanner(System.in);  
System.out.print("Enter a character: ");  
String s = input.next();  
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.

# 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.

## Example:

```
String s1 = "Hello!";
String s2 = "Hello!";
if (s1.equals(s2))
    System.out.println("The strings are equal");
else
    System.out.println("The strings are not equal");
```

# 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.

## Example:

```
String s1 = "Hello!";
String s2 = "Hello!";
if (s1.equals(s2))
    System.out.println("The strings are equal");
else
    System.out.println("The strings are not equal");
```

OrderTwoCities

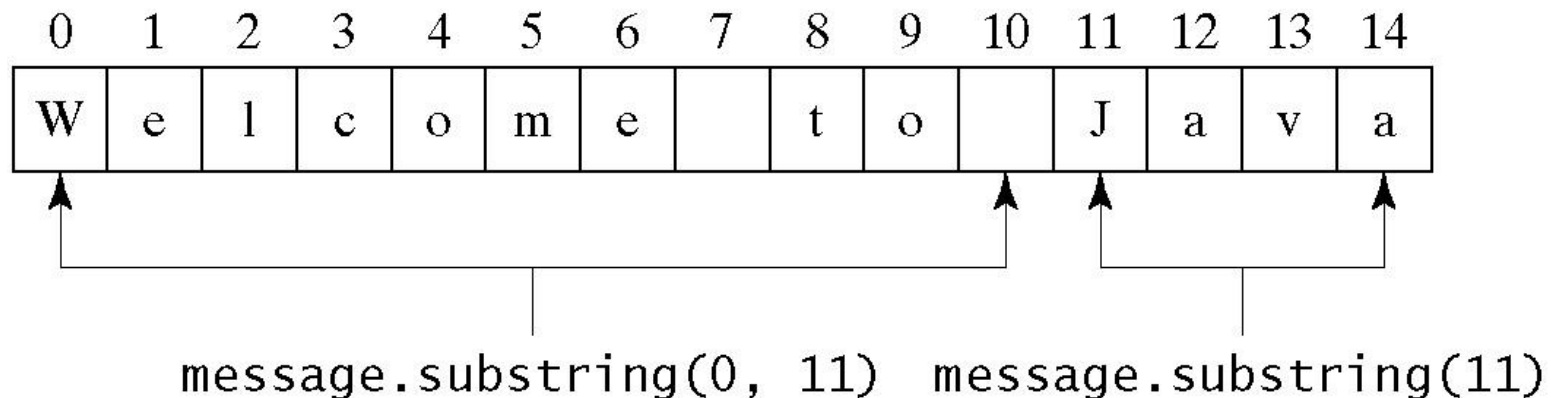
Do not use `s1 == s2` to test for equality – more later!

# 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
<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> Note that the character at <code>endIndex</code> is not part of the substring.

## Example:

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



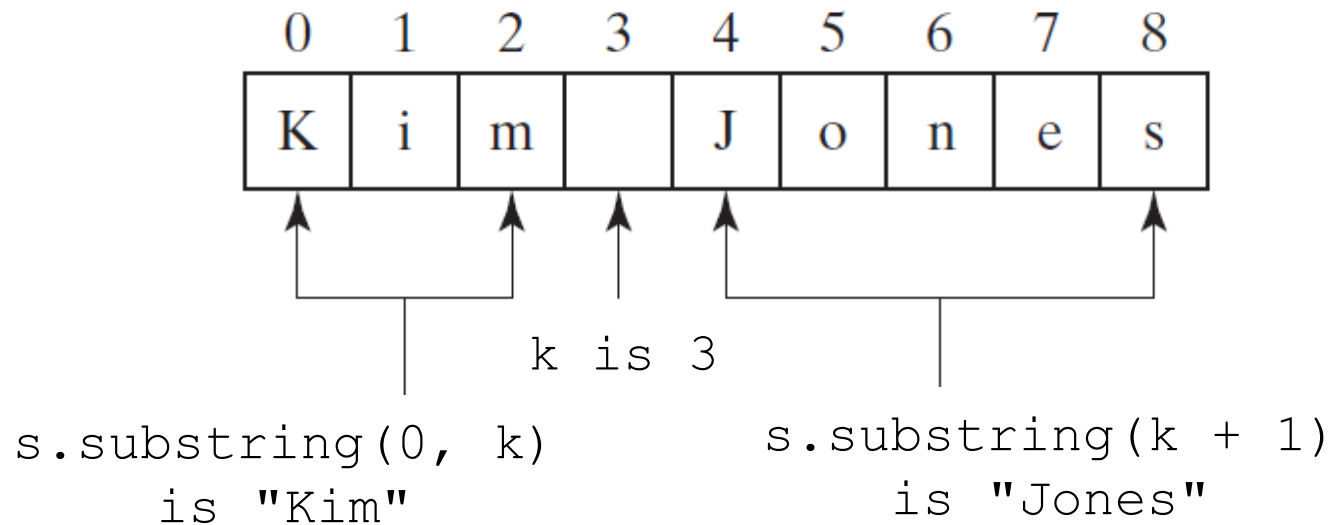
# Finding a Character or a Substring in a String

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.



# Example – Obtain First and Last Names from a String

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



# Conversion between Strings and Numbers

Converting from strings to numbers:

```
String str = "123";  
int i = Integer.parseInt(str); // i is 123
```

```
String str = "123.456";  
double d = Double.parseDouble(str); // d is 123.456
```

Converting from numbers to strings:

```
int number = 1;  
String s = "" + number; // s is "1";
```

# Converting a Hexadecimal Digit to a Decimal Value

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

HexDigit2Dec

# Formatted Output

```
int p = 10;  
double q = 2.0 / 3;
```

# Formatted Output

```
int p = 10;
```

```
double q = 2.0 / 3;
```

```
System.out.println("p is " + p + " and q is " + q);
```

```
// displays: p is 10 and q is 0.6666666666666666
```

# Formatted Output

```
int p = 10;
```

```
double q = 2.0 / 3;
```

```
System.out.println("p is " + p + " and q is " + q);
```

```
// displays: p is 10 and q is 0.6666666666666666
```

```
System.out.printf("p is %d and q is %f\n", p, q);
```

```
// displays: p is 10 and q is 0.666667
```

# Formatted Output

```
int p = 10;  
double q = 2.0 / 3;
```

```
System.out.println("p is " + p + " and q is " + q);  
// displays: p is 10 and q is 0.6666666666666666
```

```
System.out.printf("p is %d and q is %f\n", p, q);  
// displays: p is 10 and q is 0.666667
```

```
System.out.printf("p is %d and q is %.2f\n", p, q);  
// displays: p is 10 and q is 0.67
```

# Formatting Output

**TABLE 4.11** Frequently Used Format Specifiers

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



# Next Topics...

## Chapter 5

- To write programs for executing statements repeatedly using loops: `do-while`, `while`, and `for`.
- To discover the similarities and differences of three types of loop statements.
- To control a loop with a sentinel value.
- To write nested loops.
- To implement program control with `break` and `continue`.