



# **COSC 111**

## **Computer Programming I**

### **Additional slides on Mathematical Functions, Characters, and Strings**

**Dr. Firas Moosvi**

# The Math Class

# Mathematical Functions and Constants

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

- *trigonometric methods,*
- *exponent methods, and*
- *service methods*

Two useful double constants,

- **PI**
- **E** (the base of natural logarithms).

# Trigonometric Methods

**Math.sin(r)**

**Math.cos(r)**

**Math.tan(r)**

**Math.acos(r)**

**Math.asin(r)**

**Math.atan(r)**

**Math.toRadians(d)**

**Math.toDegree(r)**

## Examples:

**Math.toDegrees(Math.PI/2) returns 90.0**

**Math.toRadians(30) returns 0.5236  
(i.e.,  $\pi/6$ )**

**Math.sin(0) returns 0.0**

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

**Math.sin(Math.toRadians(90)) 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

**Math.pow(a, b)**

- returns a raised to power of b.

**Math.sqrt(a)**

- returns square root of a.

**Math.exp(a)**

- returns e raised to power of a.

**Math.log(a)**

- returns natural logarithm of a.

**Math.log10(a)**

- returns the 10-based logarithm 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 22.9176**

**Math.sqrt(4) returns 2.0**

**Math.sqrt(10.5) returns 3.24**

# Rounding Methods

## `Math.ceil(x)`

- `x` rounded up to its nearest integer.

## `Math.floor(x)`

- `x` is rounded down to its nearest integer.

## `Math rint(x)`

- `x` is rounded to its nearest integer.
- If `x` is equally close to two integers, the ***even*** one is returned

## `Math.round(x)`

- Return `(int)Math.floor(x+0.5)`.

In all methods, the result is returned as a double value.

# Rounding Methods Examples

## 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 methods

**Math.max(a, b)**

**Math.min(a, b)**

- Return the maximum or minimum of a and b.

**Math.abs(a)**

- Returns the absolute value of a.

## 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

## `random()`

- Returns a random double value in the range [0.0, 1.0).
  - $0 \leq \text{Math.random()} < 1.0$

Examples:

<code>(int)(Math.random() * 10)</code>	→	Returns a random integer between 0 and 9.
<code>50 + (int)(Math.random() * 50)</code>	→	Returns a random integer between 50 and 99.

In general,

<code>a + Math.random() * b</code>	→	Returns a random number between a and a + b, excluding a + b.
------------------------------------	---	---

# Clicker Question

What is the output? (note that Math class is imported by default)

```
double x = floor(2.3) + ceil (1.01) ;  
System.out.println(x) ;
```

- A. 3
- B. 3.0
- C. 4
- D. 4.0
- E. Error

# Clicker Question

What is the output? (note that Math class is imported by default)

```
double x = Math.floor(-2.3) + Math.ceil (1.01);  
System.out.println (x);
```

- A. -2.0
- B. -1.0
- C. 0.0
- D. 1.0
- E. Error

# Clicker Question

Which of the following statements generates a random integer from 0 to 25 inclusive?

A. `int x = Math.random() * 26;`

B. `int x = (int) Math.random() * 26;`

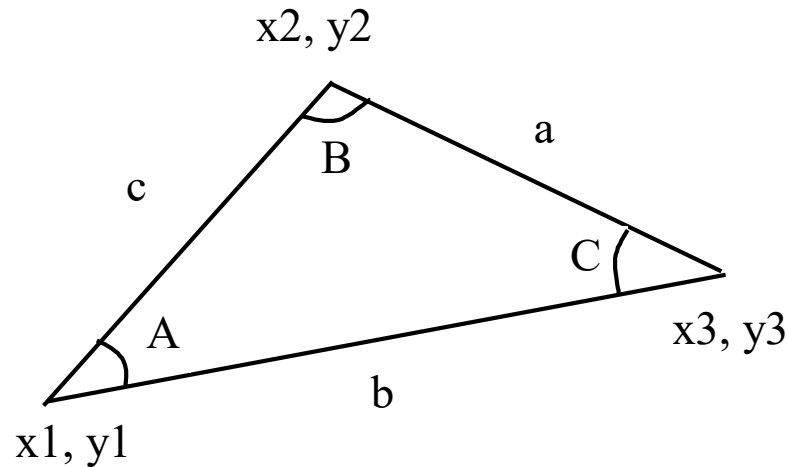
C. `int x = (int)( Math.random() * 26 );`

D. `int x = (int)( Math.random() * 25 );`

# Practice



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.



$$A = \text{acos}((a * a - b * b - c * c) / (-2 * b * c))$$

$$B = \text{acos}((b * b - a * a - c * c) / (-2 * a * c))$$

$$C = \text{acos}((c * c - b * b - a * a) / (-2 * a * b))$$

# Practice



Write a program that declares a variable `degrees`, assigns it to 30 then 60, and in both cases prints out the following **formatted** output

Degrees	Radians	Sine	Cosine	Tangent
30	0.5236	0.5000	0.8660	0.5774
60	1.0472	0.8660	0.5000	1.7321

## Algorithm

- Step 1: print out the header using `printf`
- Step 2: Declare a variable `degrees` and initialize it to 30
- Step 3: print the second line. Hint: use `printf` and Math functions
- Step 4: assign 60 to `degrees`
- Step 5: print the third line. Hint: use `printf` and Math functions

# Formatting Console Output

# Intro Example

Lets say you have 4 variables and want to print them in the format “identifier: value” on the same line as follows:

w=9    x=1    y=4    z=7

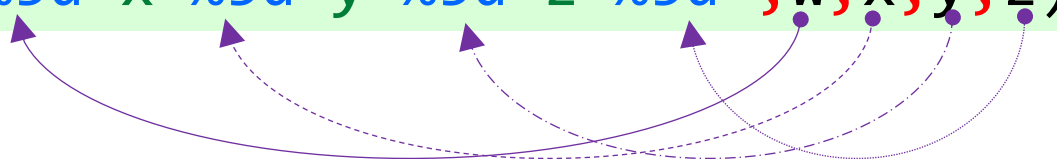
One way to do this is as follows:

```
System.out.println("w="+w+"\tx="+x+"\ty="+y+"\tz="+z);
```

One problem with above statement is that it is not easy to write (many quotes and +’s) or read (many items separated by +).

A better way to write this statement would be using printf()

```
System.out.printf("w=%5d x=%5d y=%5d z=%5d", w, x, y, z);
```



*Each value is inserted at  
corresponding placeholder*



# Formatting Console Output: `printf`

You can use the **`System.out.printf`** method to display formatted output on the console. The syntax to invoke this method is

```
System.out.printf(format, item1, ..., itemk)
```

where

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

<i>Format Specifier</i>	<i>Output</i>	<i>Example</i>
<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>%s</code>	a string	"Java is cool"

# Formatting Console Output

Example:

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



The diagram illustrates the mapping of variables to format specifiers in the printf statement. An arrow points from the variable 'count' to the format specifier '%d'. Another arrow points from the variable 'amount' to the format specifier '%f'. A bracket labeled 'items' spans the entire printf statement, indicating the scope of the formatting operation.

display

count is 5 and amount is 45.560000

# Formatting Console Output

You can specify the width and precision in a format specifier, as shown in the examples:

- %5c** Output the character and add four spaces **before** the character item, because the width is 5.
- %6b** Output the Boolean value and add one space before the false value and two spaces before the true value.
- %5d** Output the integer item with width at least 5.
- %9.2f** Output the floating-point item with width at least 9 including a decimal point and two digits after the point. Thus, there are 6 digits allocated before the decimal point.
- %8s** Output the string with width at least 8 characters.

## Notes:

- If an item requires more spaces than the specified width, the width is automatically increased.
- By default, the output is right justified. You can put the minus sign (-) in the format specifier to specify that the item is left justified
- The **%** sign denotes a format specifier. To output a literal % in the format string, use **%%**.

# Formatting Console Output: Example1

Examples:

```
System.out.printf("%8d%8s%8.1f\n", 1234, "Java", 5.63);
```

|← 8 →|← 8 →|← 8 →|  
□□□□ 1234 □□□□ Java □□□□ 5.6

```
System.out.printf("%-8d%-8s%-8.1f\n", 1234, "Java", 5.63);
```

|← 8 →|← 8 →|← 8 →|  
1234 □□□□ Java □□□□ 5.6 □□□□

# Clicker Question

What is the output?

```
int x = 7;  
System.out.println("value of x is %d" + x);
```

- A. value of x is 7
- B. value of x is %d7
- C. value of 7 is 7
- D. value of 7 is %d7
- E. Error

# Clicker Question

What is the output?

```
int x = 7;  
System.out.printf("value of x is %d" + x);
```

- A. value of x is 7
- B. value of x is %d 7
- C. value of 7 is 7
- D. value of 7 is %d 7
- E. Error

# Clicker Question

What is the output?

```
int x = 7;  
System.out.printf("value of x is %d", x);
```

- A. value of x is 7
- B. value of x is %d 7
- C. value of 7 is 7
- D. value of 7 is %d 7
- E. Error

# Clicker Question

What is the output?

```
int x = 7;  
System.out.printf("value of x is %d" + x, x);
```

- A. value of x is 77
- B. value of x is %d 77
- C. value of 7 is 77
- D. value of 7 is %d 77
- E. Error



# Character Data Type and Operations

# Character Data Type

The character data type, **char**, is used to represent a single character.

A character literal is enclosed in single quotation marks.

Examples:

```
char letter = 'A';
```

```
char numChar = '4';
```

*Unicode for character 'A' is 0041*



```
char letter = '\u0041';
```

```
char numChar = '\u0034';
```

The ++ and -- operators can be used on char variables to get the next or preceding Unicode character. For example.

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

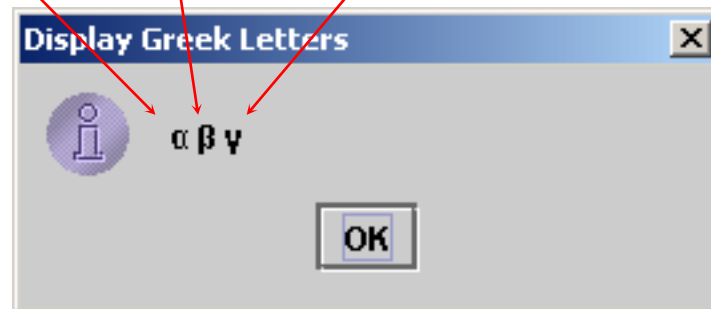
```
System.out.println(++ch); // displays character b
```

# Unicode Format

Java characters use Unicode, a 16-bit encoding scheme 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'`.

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



# Escape Sequences for Special Characters

A character preceded by a backslash (\) is an escape sequence and has special meaning to the compiler.

<i>Escape Sequence</i>	<i>Name</i>
<code>\b</code>	Backspace
<code>\t</code>	Tab
<code>\n</code>	Linefeed
<code>\f</code>	Formfeed
<code>\r</code>	Carriage Return
<code>\\</code>	Backslash
<code>\"</code>	Double Quote

Example: `System.out.println("Welcome to \"UBC\"");`

The output is: `Welcome to "UBC"`

# Casting between char and Numeric Types

A char can be cast into numeric types, and vice versa.

```
int i = 'A';      // decimal value of A which is 65 is stored in i
```

```
int i = (int) 'A'; // Same as above
```

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

floating-point values (must be explicit)

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

# Comparing and Testing Characters

Characters can be compared based on their Unicode values.

## Examples:

**'1' < '8'**

- **True** because the Unicode for **'1'** (**49**) is less than the Unicode for **'8'** (**56**).

**'a' < 'b'**

- **True** because the Unicode for **'a'** (**97**) is less than the Unicode for **'b'** (**98**).

**'a' < 'A'**

- **False** because the Unicode for **'a'** (**97**) is greater than the Unicode for **'A'** (**65**).

# Methods in the Character Class

**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.

# Methods in the Character Class

For example,

- `Character.isDigit('a')` returns `false`
- `Character.isLetter('a')` returns `true`
- `Character.isLowerCase('a')` returns `true`
- `Character.isUpperCase('a')` returns `false`
- `Character.toLowerCase('T')` returns `t`
- `Character.toUpperCase('q')` returns `Q`

Q. How to convert a character '3' to integer 3?

**Answer:** subtract '0'

```
char ch = '3';
```

```
int x = ch - '0'; // x = 3
```



# Clicker Question

What is the output?

```
char x = 'a', y = 'c';  
System.out.print(++x);  
System.out.print(y++);  
System.out.print(y - x);
```

A. ac2

B. bc2

C. bd2

D. ac1

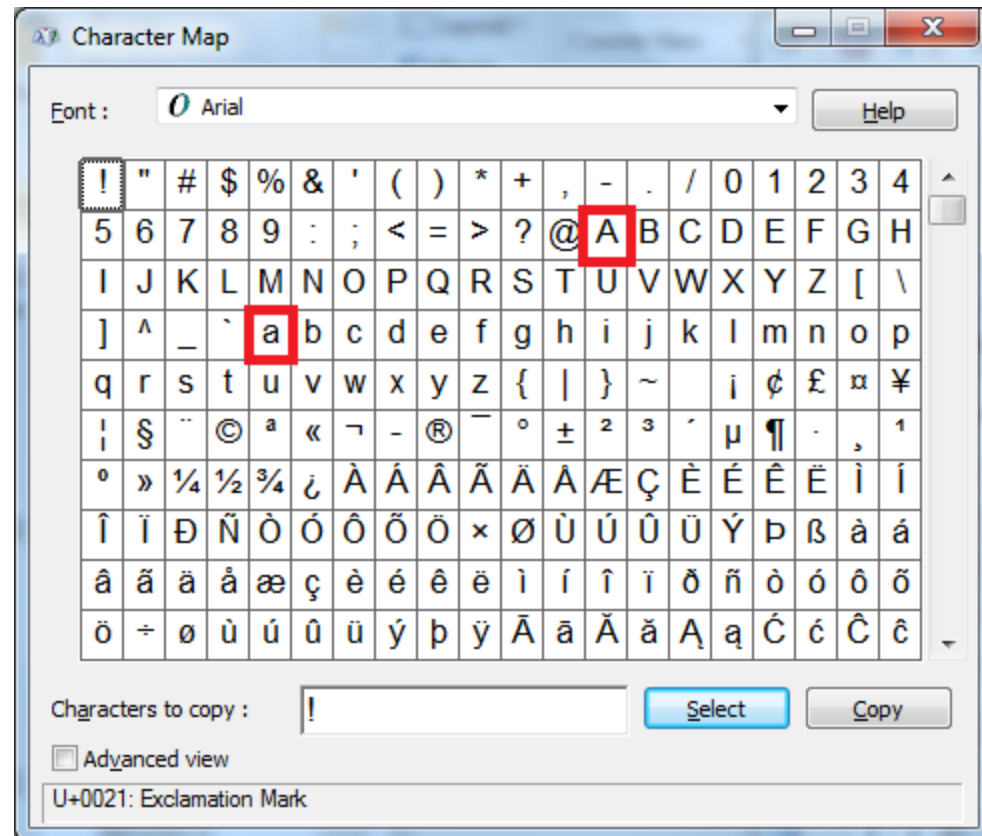
E. bc1

# Clicker Question

What is the output?

```
System.out.print('a' < 'b');  
System.out.print('a' <= 'A');  
System.out.print('a' > 'b');  
System.out.print('a' != 'b');
```

- A. true true true true
- B. false false false false
- C. true true false true
- D. true false false true
- E. Error



# Clicker Question

What is the output?

```
char ch = '5';
```

```
int x = ch - '0'; //converts from '5' to 5 ★
```

```
System.out.println(x + 2);
```

- A. 52
- B. 502
- C. 7
- D. Error

# Clicker Question

What is the output?

```
char ch = '33';
```

```
System.out.println(ch + 2);
```

A. 332

B. 35

C. "35"

D. An number other than 35 and 332

E. An error

# The String Type

# The String Type

To represent *a sequence of characters*, use the data type called **String**.

String message = "Welcome to Java";

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

The String type is **not a primitive type**.

- *String* is actually a predefined class in the Java library just like the *System* class and *Scanner* class. 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,
  - 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.

# Methods for String Objects

## **s1.length()**

- Returns the number of characters in the string s1.

"Welcome".length() returns 7

## **s1.charAt(index)**

- Returns the character at the specified index from string s1.

"Welcome".charAt(0) returns 'W'

## **String s2 = s1.toUpperCase()**

- Returns a new string s2 with all letters of s1 in uppercase.

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

## **String s2 = s1.toLowerCase()**

- Returns a new string with all letters in lowercase.

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

## **s1.trim()**

- Trims whitespace characters on both sides of s1.

" Welcome to COSC111 ".trim() returns a new string,  
"Welcome to COSC111"

# Methods for String Objects

Strings are objects in Java.

•

The methods in the previous slide can only be invoked from a **specific string instance**. For this reason, these methods are called instance methods. e.g,

```
String s = "abc";  
int x = s.length();
```

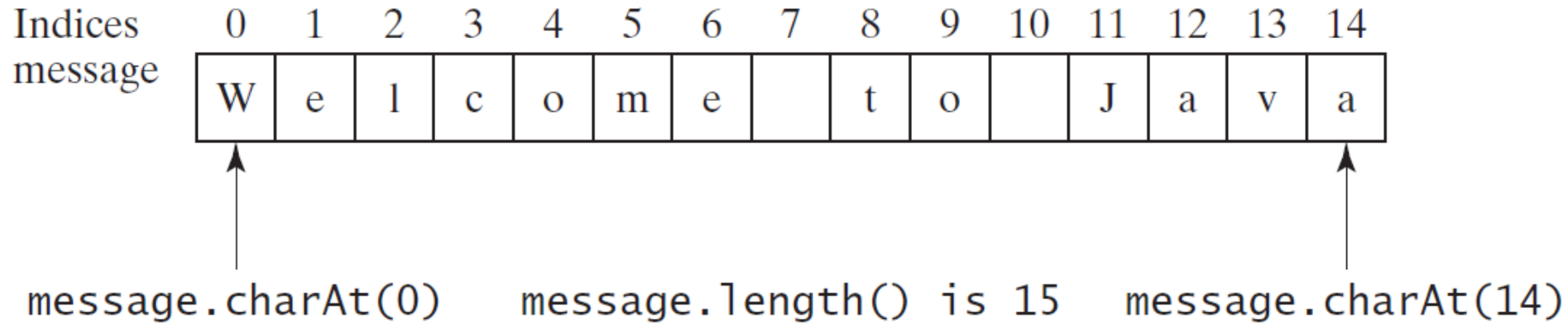
A non-instance method is called a static method. A static method can be invoked without using an object.

- All the methods defined in the Math class are static methods. They are not tied to a specific object instance. They can be invoked directly using the Math class. e.g.,

```
Math.sin(Math.PI/2);
```



# String: charAt ( )



```
String s= "Welcome to Java";  
System.out.println("First character is "  
                    + s.charAt(0));
```

# Reading a String from the Console

You can use a Scanner object to read a string from the console.

You may use the methods:

- `next()`.
  - To reads a 'token'.
  
- `nextLine()`.
  - To read a line of text (ends with *newline* character)
  - The newline character is not read.

# Reading a String from the Console

## Using next() method.

```
import java.util.Scanner;
public class Ex1 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter 2 words separated by spaces: ");
        String s1 = input.next();
        String s2 = input.next();
        System.out.println("s1 is " + s1);
        System.out.println("s2 is " + s2);
    }
}
```

## Using nextLine() method

```
import java.util.Scanner;
public class Ex1 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter a line: ");
        String s = input.nextLine();
        System.out.println("The line entered is " + s);
    }
}
```

# Reading a Character from the Console

use the **nextLine()** method to read text and then invoke the **charAt(0)** method on the text to return the first character.

```
import java.util.Scanner;
public class Ex1 {
    public static void main(String[] args) {
        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);
    }
}
```

# Practice

Write a program that reads from a user a single letter representing a hexadecimal digit and convert it into a decimal value. Hexadecimal digits represented by a letter are (with their values): A = 10, B=11, ..., F=15.

- Your program should accept lower case values
  - e.g. both A and a are converted to 10
- Hint: the '-' operator can be used with characters.

```
import java.util.Scanner;
public class HexToDec {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter a letter representing a hex digit (A to F): ");
        String text = in.nextLine();
        char ch = text.charAt(0);
        ch = Character.toUpperCase(ch);
        int dec = ch - 'A' + 10;
        System.out.println("Decimal value is: " + dec);
    }
}
```

# More methods: Comparing Strings



**s1.equals(s2)**

- returns *true* if s1 is equal to s2

**s1.equalsIgnoreCase(s2)**

- same as equals but it is case insensitive.

**s1.compareTo(s2)**

- returns an *integer*  $> 0$ ,  $= 0$ , or  $< 0$  to indicate whether s1 is greater than, equal to, or less than s2.

**s1.compareToIgnoreCase(s2)**

- same as compareTo except that it is case insensitive

**s1.startsWith(prefix)**

- returns *true* if s1 starts with the specified prefix.

**s1.endsWith(suffix)**

- Returns true if s1 ends with the specified suffix.

# compareTo ( )

The method returns

- **0** if **s1** is equal to **s2**
- **Negative value** if **s1** is lexicographically less than **s2**, and
- **Positive value** if **s1** is lexicographically greater than **s2**.

The value returned from the **compareTo** method depends on the **offset of the first two different characters** in **s1** and **s2** from left to right.

Example:

**Assume s1 is “abc” and s2 is “abe”**

**s1.compareTo(s2) returns -2.**

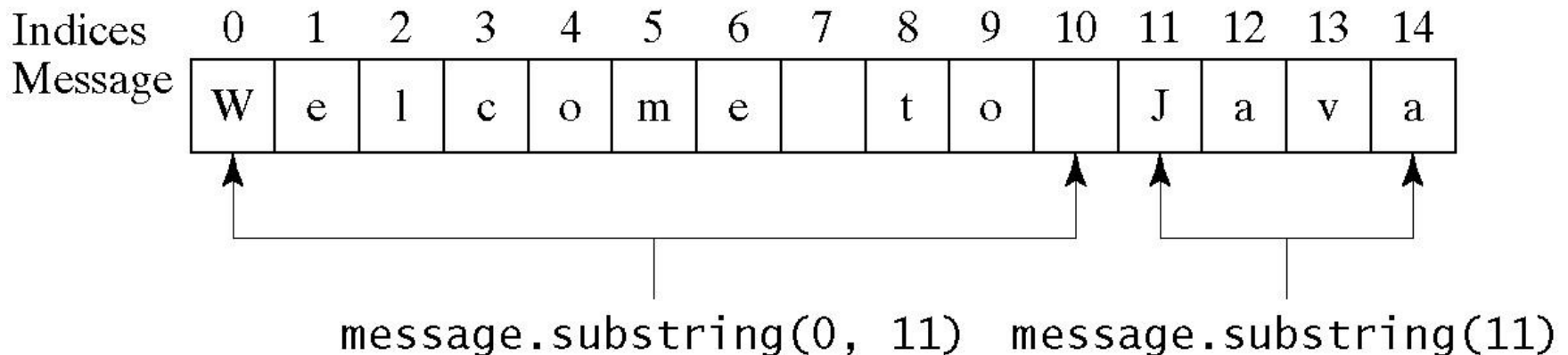
# substring()

## substring(beginIndex)

- Returns this string's substring that begins with the character at the specified beginIndex and extends to the end of the string.

## substring(beginIndex, endIndex)

- Returns this string's substring that begins at the specified beginIndex and extends to the character at index endIndex – 1,. Note that the character at endIndex **is not part of** the substring.





# `indexOf()` and `lastIndexOf()`

**`s1.indexOf(s)`**

- Returns index of the first occurrence of `s` in the `s1`.

**`s1.indexOf(s, idx)`**

- Returns index of the first occurrence of `s` after `idx` in `s1`.

**`s1.lastIndexOf(s)`**

- Returns index of the last occurrence of `s` in `s1`.

**`s1.lastIndexOf(s, idx)`**

- Returns index of `s`, searching backward starting at `idx`.

**All above methods**

- **return -1 if no match is found.**
- `s` could be a character or a string

# Example1: Finding a Character/Substring

`"Welcome to Java".indexOf('W')` returns 0.

`"Welcome to Java".indexOf('o')` returns 4.

`"Welcome to Java".indexOf('o', 5)` returns 9.

`"Welcome to Java".indexOf("come")` returns 3.

`"Welcome to Java".indexOf("Java", 5)` returns 11.

`"Welcome to Java".indexOf("java", 5)` returns -1.

`"Welcome to Java".lastIndexOf('W')` returns 0.

`"Welcome to Java".lastIndexOf('o')` returns 9.

`"Welcome to Java".lastIndexOf('o', 5)` returns 4.

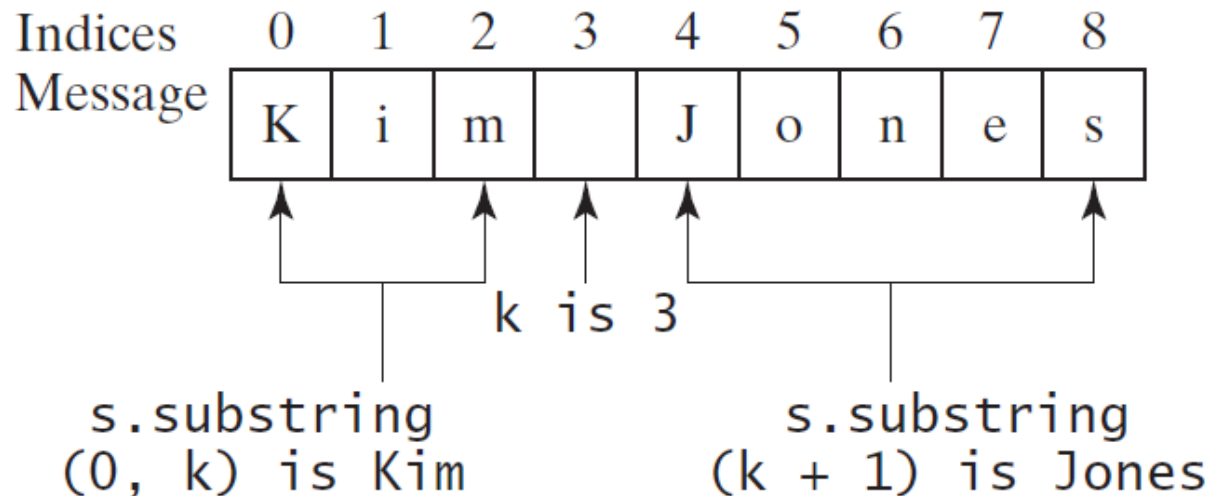
`"Welcome to Java".lastIndexOf("come")` returns 3.

`"Welcome to Java".lastIndexOf("Java", 5)` returns -1.

`"Welcome to Java".lastIndexOf("Java")` returns 11.

# Example2: Extracting two words from a string

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



# Practice

Given the following strings,

```
String s1 = "Welcome to Java";
```

```
String s2 = "Programming is fun";
```

```
String s3 = "Welcome to Java";
```

What are the results of the following expressions?

(c) `s1.equals(s2)`

(d) `s1.equals(s3)`

(e) `s1.compareTo(s2)`

(f) `s2.compareTo(s3)`

(g) `s2.compareTo(s2)`

(h) `s1.charAt(0)`

(i) `s1.indexOf('j')`

(j) `s1.indexOf("to")`

(k) `s1.lastIndexOf('a')`

(l) `s1.lastIndexOf("o", 15)`

(m) `s1.length()`

(n) `s1.substring(5)`

(o) `s1.substring(5, 11)`

(p) `s1.startsWith("Wel")`

(q) `s1.endsWith("Java")`

(r) `s1.toLowerCase()`

(s) `s1.toUpperCase()`

(t) `s1.concat(s2)`

(u) `s1.contains(s2)`

(v) `"\t Wel \t".trim()`

# Clicker Question

What is the output?

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

```
String s1 = in.nextLine(); // User enters abc
```

```
String s2 = in.nextLine(); // User enters abc
```

```
System.out.println(s1.equals(s2));
```

```
System.out.println(s1 == s2);
```

- A. true      true
- B. false     false
- C. true      false
- D. false     true

# Conversion between Strings and Numbers

## Numeric String → Number :

```
int x = Integer.parseInt("5");           //x is now equal to 5  
double y=Double.parseDouble("5.2"); //y is now equal to 5.2
```

## Number → string:

```
String s = 3.1 + "";
```

//s is now equal to "3.1"