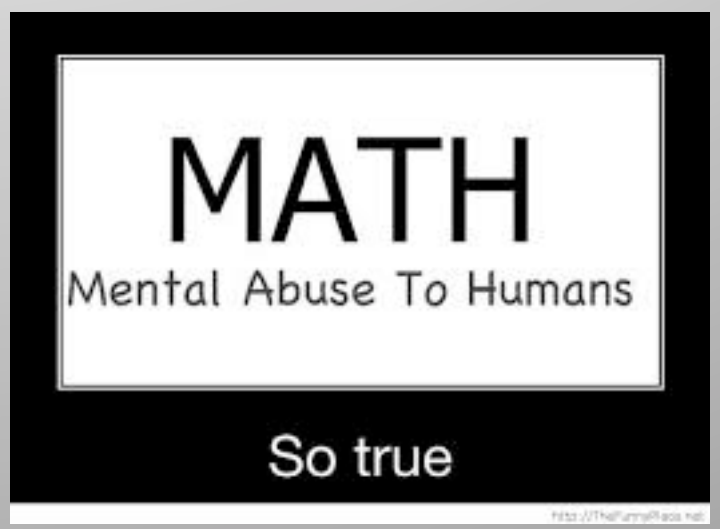
Mathematical Functions, Characters

Reading: Chapter 4, 4.1 - 4.3

The Math Class

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

- There are also several very useful constants available in the Math class.



Constants from the Math class

- Two constants: One for the value of pi and the other for the value of *e* (the base of the natural logarithm).
- Both are doubles.
- No parentheses come after the constants.

`Math.PI`

3.141592653589793238 46

`Math.E`

2.718281828459045

Exponent Methods

Method	*Description*
`exp(x)`	Returns e raised to power of x (e^x).
`log(x)`	Returns the natural logarithm of x ($\ln(x) = \log_e(x)$).
`log10(x)`	Returns the base 10 logarithm of x ($\log_{10}(x)$).
`pow(a, b)`	Returns a raised to the power of b (a^b).
`sqrt(x)`	Returns the square root of x ($\sqrt{x}$) for x >= 0.

```
Math.exp(1)
```

e^1

returns 2.71828

```
Math.sqrt(4)
```

$\sqrt{4}$

returns 2.0

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

2^3

returns 8.0

Rounding Methods

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

`Math.ceil(2.8)`

returns 3.0

`Math.floor(2.1)`

returns 2.0

`Math.rint(2.9)`

returns 3.0

`Math.round(2.6)`

returns 3

`Math.ceil(2.1)`

returns 3.0

`Math.floor(2.8)`

returns 2.0

`Math.rint(2.1)`

returns 2.0

`Math.round(2.1)`

returns 2

min, max and abs Methods

```
Math.min(2, 5)
```

Finds the minimum of two numbers

returns 2

```
Math.max(2, 5)
```

Finds the maximum of two numbers

returns 5

```
Math.abs(-4)
```

Finds the absolute value of a number

returns 4

```
Math.min(2.1, 1.9)
```

Finds the minimum of two numbers

returns 1.9

```
Math.max(2.1, 1.9)
```

Finds the maximum of two numbers

returns 2.1

```
Math.abs(-4.3)
```

Finds the absolute value of a number

returns 4.3

Example

Write a program that asks the user to enter two points and then computes the distance between them.

distance = $\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$

```
Enter x1: 1
Enter y1: 2
Enter x2: 4
Enter y2: 6
The distance is 5.0.
```

We are going to have to use two methods from the `Math` class. What are they?

`sqrt` and `pow`

```
import java.util.Scanner;

public class Distance
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        double x1, y1, x2, y2;

        S.O.P("Enter x1: ");
        x1 = keyboard.nextDouble();
        S.O.P("Enter x2: ");
        x2 = keyboard.nextDouble();
        S.O.P("Enter x1: ");
        y1 = keyboard.nextDouble();
        S.O.P("Enter x2: ");
        y2 = keyboard.nextDouble();

        double diffx = x2 - x1;
        double diffy = y2 - y1;
        double inside = Math.pow(diffx, 2) + Math.pow(diffy, 2);
        double distance = Math.sqrt(inside);

        S.O.P.L("The distance is " + distance + ".");
    }
}
```

Character Data Type

- A character data type represents a single character.
- Computers use binary numbers (0's and 1's) to store character data types.
- Mapping a character to its binary representation is called encoding.
- Java uses Unicode, an encoding scheme.
- Unicode was originally designed as a 16-bit character encoding. However, it turned out that the 65,536 characters possible using 16-bit encoding was not sufficient.
- It's been extended to allow up to 1,112,064 characters.
- The characters that go beyond the original 16-bit limit are called supplementary characters.

Character Data Type

- A 16-bit Unicode character takes two bytes.
- Preceded by \u and followed by 4 hexadecimal digits that run from \u0000 to \uFFFF

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

char is the data type →

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

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

Assigns the letter A to the variable letter using Unicode

Comparing Characters

- Two characters can be compared using the relational operators just like comparing two numbers.
- When you compare two characters, the Unicode values are compared.
- The order is 0 - 9, A - Z, a - z
- What about the number 10? That's not a single character!

```
S.O.P.L('a' < 'b');
S.O.P.L('a' < 'A');
S.O.P.L('1' < '8');
```

True. Unicode for 'a' (97) is less than Unicode for 'b' (98)

False. Unicode for 'a' (97) is greater than Unicode for 'A' (65)

True. Unicode for '1' (49) is less than Unicode for '8' (56)

Why compare characters?

Assume that the variable letter has been declared and has a value assigned to it.

```
if (letter >= ‘A’ && letter <= ‘Z’)
    S.O.P.L(letter + “ is uppercase.”);
if (letter >= ‘a’ && letter <= ‘z’)
    S.O.P.L(letter + “ is lowercase.”);
if (letter >= ‘0’ && letter <= ‘9’)
    S.O.P.L(letter + “ is numeric.”);
```

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.

```
S.O.P.L("Character 6 is a digit: " + Character.isDigit('6'));

S.O.P.L("Character a is lower case: " + Character.isLowerCase('a'));

S.O.P.L("Character A is upper case: " + Character.isUpperCase('A'));
```

```
char letter = ‘a’;
char upper = Character.toUpperCase(letter);
S.O.P.L(“The letter ” + letter + “ capitalized is “ + upper);
```

Output Window

```
The letter a capitalized is A
```

Escape Sequences for Special Characters

What if you wanted to print out the following statement:

He said “Good morning!”

```
S.O.P.L(“He said “Good morning!””);
```

Compile error!

- The compiler thinks the second quotation character is the end of the string.
- It does not know what to do with the rest of the characters.
- You have to “escape” certain special characters when printing.

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

```
S.O.P.L("He said "Good morning!"");
```

Wrong!

```
S.O.P.L("He said \"Good morning!\"");
```

Right!

What if…

```
S.O.P.L('a');
S.O.P.L('b');
S.O.P.L('a' + 'b');
```

```
Output Window

a
b
195
```

When you use the + operator with character data types it returns the decimal code (which it gets from the hexadecimal code) and adds them together.

No, you do not have to memorize all the decimal values for the characters!!