

**HELLO WORLD!**  
**I'M ALIVE!**



CS 0007  
Introduction to  
Computer Programming

Luís Oliveira

Summer 2020

# HELLO!

# Hello world!

```
/*  
    Author: Luis Oliveira  
    This is a simple example of a Java program  
*/  
public class Hello  
{  
    public static void main( String[] args )  
    {  
        // This is the code that will run  
        System.out.print("Hello World!");  
    }  
}
```

# Hello world! - Decrypted

```
/*  
    This is a block comment. It starts with the “forward-slash asterisk”  
    Nothing you write here is seen by Java and it’s compiler.  
    It ends with the “forward-slash asterisk”, again  
*/  
public class Hello ← This is the class header Hello is it’s name  
{  
    public static void main( String[] args ) ← This is where your program starts!  
                                                It’s the main function header  
    {  
        // This is an in-line comment. Next line is seen by Java!  
        System.out.print("Hello World!");  
    }  
}
```

# ABOUT VARIABLES

# Primitive numeric variable types

**byte** - stores tiny integer numbers  
range: -128 → 127

**short** - stores small integer numbers  
range: -32,768 → 32,767

**int** - stores integer numbers  
range: -2,147,483,648 → 2,147,483,647

**long** - stores large integer numbers  
range:  
-9,223,372,036,854,775,808 → 9,223,372,036,854,775,807

**float, double** - store real numbers → double has more range and precision (more decimal places)

float range: 1.401e-45 to 3.402e38 (same negative)

double range: 4.941e-324 to 1.798e308 (same negative)  
→ it's complicated 😊

Declaring a variable:  
**type** name = value;

# Primitive non-numerical variable types (and String)



- char** (like in charizard!)- stores text characters  
e.g.: A single letter, or a single punctuation mark
- String** - stores a text i.e, a bunch of **chars**  
variable size!
- boolean** - truthiness, i.e. true or false  
range: ermm... either **true** or **false**
- others?** - we can create types! But we'll discuss that later

# Literals

- When you *type* a number or string, that's a literal.
  - Only primitive types and String have literals
    - String is special because it's VERY common.

• E.g.

type	literal
String	"Hello Luis!"
char	'X'
boolean	false
int	42
double	3.14159

Note: Strings use double quotes, chars use single quotes!

```
String text = "Hello Luis!";  
char letter = 'X';  
boolean validPoint = true;  
int number = 42;  
double pi = 3.14159;
```



# Naming rules

- Variables

- Names must start with a letter or a \_ (underscore)
- Names can contain numbers
  - E.g: age, \_age, part1, \_variable
- Names are all low-case, except to separate different words
  - E.g.: word, twoWords, threeWordVariable
- Names are case sensitive: **variable** is not the same as **vArlaBLE**
- Use good names!
  - Bad names: a, aa, aaa, abc, here, qwerty
    - I've seen this before: x, xx, xxx; ← Don't!
  - Good names: age, height, position, distance, sumOfVariables

# Naming rules (cont.)

- Constants

- Names must start with a letter or a \_ (underscore)
- Names can contain numbers
  - E.g: age, \_age, part1, \_variable
- Names are all upper-case
  - E.g.: WORD, TWO\_WORDS, MULTIPLE\_WORD\_CONSTANT
- Use good names!
- Use the keyword **final**
  - E.g. **final** **int** INCHES\_IN\_A\_FOOT = 12;

# Operations on variables

- Assignments

- = → The assignment operator (doesn't compare)  
e.g.: **destination** = **source**
- First calculate EVERYTHING to its right (variable or expression)
- Finally store the result into the variable to its left

- Examples:

```
age = 33;           // age gets the number 33
age = age - 1;      // age value is changed to 32 : the old
                    //          value of age minus 1
halfAge = age / 2;   // halfAge gets 16: the value of age divided by 2
```

# Java numeric operators (easy)

Operator	Name	Type	Example
-	Negation	Unary	result = -b;
*	Multiplication	Binary	result = a * b;
/	Division	Binary	result = a / b;
%	Modulus	Binary	result = a % b;
+	Addition	Binary	result = a + b;
-	Subtraction	Binary	result = a - b;

# Java relational operators (medium)

Operator	Name	Type	Example
<b>== (don't confuse with single = )</b>	Equals	Binary	a == b;
<b>!=</b>	Not equal	Binary	a != b;
<b>&gt;</b>	Greater than	Binary	a > b;
<b>&gt;=</b>	Greater than or equal	Binary	a >= b;
<b>&lt;</b>	Less than	Binary	a < b;
<b>&lt;=</b>	Less than or equal	Binary	a <= b;

# Java precedence of operators

- What happens first?

- People: Hate maths, love solving maths problems on Facebook  $\neg\_(\text{ツ})\_/-$

First	Operator	Associativity
	- (negation)	Right to left
	* / %	Left to right
Last	+ -	Left to right

Expression	Result
$2 + 10 * -2$	-18
$2 + 10 * 55 / 10$	57
$2 + 55 / 10 * 10$	52
$72 / 60 + 72 \% 60$	13
$15 * 10 \% 2 + 10$	10

- When in doubt ☺

- Parentheses
- $2 + 10 * (55 / 10)$  is the same as  $2 + (55 / 10) * 10$

# Apples and Oranges

- Integer types smaller than int, are converted to int :)
  - `aByte + 10` **is an int**
  - `aByte + aByte` **is an int**
  - `aByte + aShort` **is an int**
  - `aShort + aShort` **is an int**
- Types larger than int keep their type
  - E.g., `aByte + aLong` **is a long**
- Real numbers turn into the more precise type in expression
  - E.g., `aDouble/aFloat` **is a double**
- Operations with Strings, become strings
  - `"The number is: " + anInt` **is a String**

# Shrinking values (aka casts :)

- Casts allow us to fit a LARGE type into a small type
  - But with great power...

```
anInt = 100;  
aByte = (byte)anInt;
```

OK!

```
anInt = 200;  
aByte = (byte)anInt;
```

DOESN'T FIT!



# NUMBERS AND BINARY

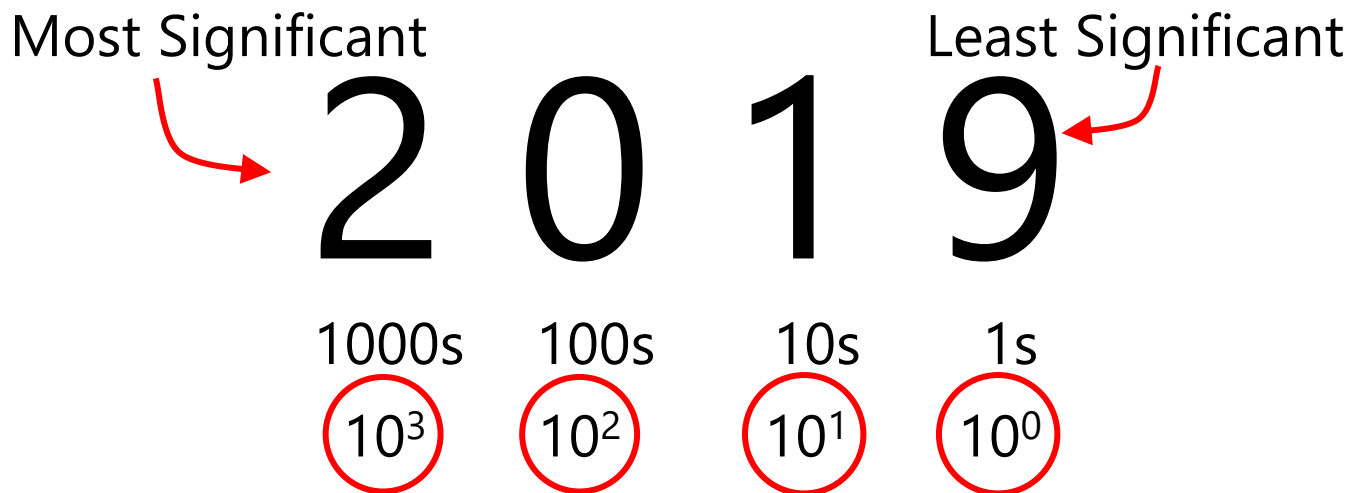
# Positional number systems

- The numbers we use are written positionally: the position of a digit within the number has a meaning.

$$\begin{array}{ccccccc} 2 & 0 & 1 & 9 & = & 2 & 0 & 0 & 0 & = & 2 & \times & 10^3 \\ & & & & & & 0 & 0 & 0 & & 0 & \times & 10^2 \\ & & & & & & & 1 & 0 & & 1 & \times & 10^1 \\ & & & & & & & & 9 & & 9 & \times & 10^0 \end{array}$$

# Positional number systems

- The numbers we use are written positionally: the position of a digit within the number has a meaning.



- How many (digits) **symbols** do we have in our number system?
  - 10: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

# Range of numbers

Suppose we have a 4-digit numeric display.

- What is the smallest number it can show?
- What is the biggest number it can show?
- How many *different* numbers can it show?
  - $9999 - 0 + 1 = 10,000$
  - What power of 10 is 10,000?
    - $10^4$



# BINARY – BASE 2

How many symbols in binary????

2

# Binary (base-2)

- We call a Binary digit a bit – a single 1 or 0
- When we say an n-bit number, we mean one with n binary digits

MSB

LSB

1001 0110 =

$2^7$   $2^6$   $2^5$   $2^4$   $2^3$   $2^2$   $2^1$   $2^0$

128s 64s 32s 16s 8s 4s 2s 1s

**To convert binary to decimal:** ignore 0s, add up place values wherever you see a 1.

$1 \times 128 +$

$0 \times 64 +$

$0 \times 32 +$

$1 \times 16 +$

$0 \times 8 +$

$1 \times 4 +$

$1 \times 2 +$

$0 \times 1$

$= 150_{10}$

- A **bit** is one binary digit, and its unit is **lowercase b**.
- A **byte** is an 8-bit value, and its unit is **UPPERCASE B**.
- A **nibble** (also nybble) is 4 bits – half of a byte
  - Corresponds nicely to a single hex digit.
- When we say "32-bit CPU," we mean it was built to use 32-bit numbers.
  - This means it can, for example, add two 32-bit numbers at once.



# Round numbers

Decimal	Binary
$10^0 = 1$	$2^0 = 1$
$10^1 = 10$	$2^1 = 2$
$10^2 = 100$	$2^2 = 4$
$10^3 = 1000$	$2^3 = 8$
$10^4 = 10000$	$2^4 = 16$
$10^5 = 100000$	$2^5 = 32$
$10^6 = 1000000$	$2^6 = 64$
$10^7 = 10000000$	$2^7 = 128$
$10^8 = 100000000$	$2^8 = 256$
$10^8 = 1000000000000$	$2^9 = 512$
$10^8 = 10000000000000000$	$2^{10} = 1024$

**byte** – 1 Byte (8 bits)  
range: -128 → 127

if 8 digits can represent  
numbers up to 99999999

8 bits can represent numbers  
up to:      11111111 <- in binary  
                 255            <- in decimal

But because we need to  
represent negative numbers  
we need to split the range  
in half.

# Primitive numeric variable types

**byte** - 1 Byte (8 bits)  
range: -128 → 127

**short** - 2 Bytes (16 bits)  
range: -32,768 → 32,767

**int** - 4 Bytes (32 bits)  
range: -2,147,483,648 → 2,147,483,647

**long** - 8 Bytes (64 bits)  
range:  
-9,223,372,036,854,775,808 → 9,223,372,036,854,775,807

**float** - 4 Bytes (32 bits)  
range: still complicated 😊

**double** - 8 Bytes (64 bits)  
range: still complicated 😊

# Primitive non-numerical variable types (and String)



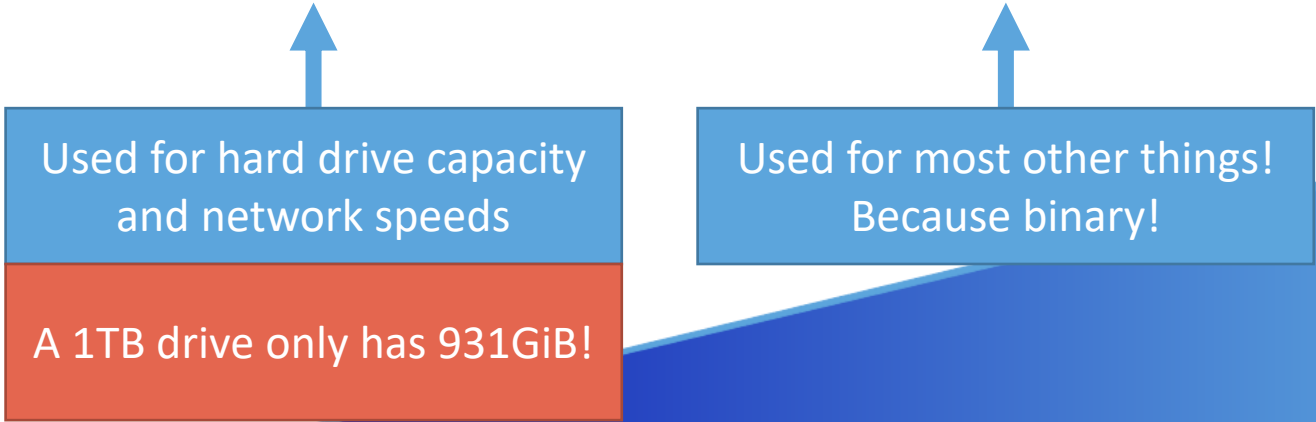
**char** (like in charizard!)- stores text characters  
e.g.: A single letter, or a single punctuation mark

**String** - stores a text i.e, a bunch of **chars**  
variable size!

**boolean** - truthiness, i.e. true or false  
range: ermm... either **true** or **false**

**others?** - we can create types! But we'll discuss that later

Potatoes	Bytes	Bytes
1g (gram)	1B (Byte)	1B (Byte)
1kg (Kilogram) = 1000g	1kB (Kilobyte) = 1000B	1kiB (Kibibyte) = 1024B (power of 2 nearest to 1000)
1Mg (Megagram) = 1000Kg	1MB (Megabyte) = 1000kB	1MiB (Mebibyte) = 1024kiB
1Gg (Gigagram) = 1000Mg	1GB (Gigabyte) = 1000MB	1GiB (Gibibyte) = 1024MiB
1Tg (Teragram) = 1000Gg	1TB (Terabyte) = 1000GB	1TiB (Tebibyte) = 1024GiB
1Eg (Exagram) = 1000Tg	1EB (Exabyte) = 1000TB	1EiB (Exbibyte) = 1024TiB



Used for hard drive capacity  
and network speeds

A 1TB drive only has 931GiB!

Used for most other things!  
Because binary!

# THE REAL WORLD IS CONFUSING!!!!

Capacity:

1000067821568 bytes

931 GB



Drive A:

Disk Clean

931GB

931GiB

Sometimes this is used to  
mean 931GiB ☹️

This always means 931GiB!  
😊

# Why binary? *Whynary?*

- Why indeed?



- What color is this?



# Why binary? *Whynary?*

- Why indeed?



- What color is this?



# Everything in a computer is a number

- Java strings are encoded using UTF-16
  - Most letters and numbers in the English alphabet are  $< 128$ .
  - “Strings are numbers”
    - 83 116 114 105 110 103 115 32 97 114 101 32 110 117 109 98 101 114 115 0

Do try this at home: what does this mean?

- 71 111 111 100 32 74 111 98 0



# EVERYTHING

- Images and colors? Numbers!
- Videos? Numbers!

