

Computer Science 101: Introduction to Java and Algorithms

A pineapple with a green crown of leaves and a brown, textured body is wearing a pair of bright yellow sunglasses. The sunglasses have dark lenses that reflect the surrounding environment. The pineapple is centered in the frame.

Section 1: The Fundamentals

Erudition Labs

Computer Science 101: Introduction to Java and
Algorithms

April 23, 2019

Contents

1	Pre-Chapter	1
1.1	What is a Program?	1
1.2	The Structure of a Java program	4
2	Basic Syntax (Video Series Lecture 1 and 2)	5
2.1	Keywords	5
3	Variables and Types (Video Series Lecture 3 and 4)	7
3.1	Java Types (Primitive Types and Strings)	7
3.2	Variables	9
4	Math in Code (Video Series Lecture 5)	10

1 Pre-Chapter

Pre-Chapters will be optional prerequisites that we think could potentially be useful for the particular Video Series Section (which we use interchangeably with Chapter). Also, throughout these pages, I will talk about origins and things I just find interesting about these topics. If none of this stuff makes sense, don't worry, it's optional and you can still learn to code and be great at it without this information. Finally, throughout these documents, we will be going over what was covered in the video series as well as go more into depth in some areas and provide additional examples.

1.1 What is a Program?

If you would like to just skip to the more useful stuff about what a program is, look for the next bold face paragraph. If you are an absolute beginner to programming or even just the tech world, you may be asking yourself, what is a program and what is programming? To many people computers are a magical black box that just does stuff when they point and click. You, however, have taken the next step and opening up this box to better understand the technology that is commanding our day and age.

You see, computers really just reduce down to physics. We can thank the 19th Century Philosopher, Logician and Mathematician, George Boole along with the Potato Famine for all the heavy lifting needed to create the foundations for the Information Age that we currently live. He invented Boolean Algebra while trying to solve the potato famine and attempt to prove the existence of God. In doing so, he created the mathematics that would become the foundations for digital logic long before we ever had the technology to utilize it.

What does any of this have to do with what a program is? Well I just think the inspirational story of George Boole is really underrated and deserves more credit. Anyways, a computer is simply made up of hardware (circuitry) that follows this Boolean Logic. We call the digital logic circuitry, Logic Gates. They perform basic logic operations such as AND, OR and XOR (exclusive OR), much in the same way as you and I do everyday. Although I will not be going into the Logic Gates, we will talk about some of the Logical Operators later. So now, a computer is simply a collection of these Logic Gates. My main point here is that computers are not magical, they follow rigorous logic given to us by circuits and thus physics. Everything boils down to an electron moving through a wire.

Luckily, you don't really need to know about any of this stuff to start programming. We have had many years to abstract away this complexity to make it easier to use. An interesting question is, how do we go from software to all the electrons moving through wires? Well, we came up with a mathematical abstraction to represent electron flow in wires called binary. The basic idea (and pretty over simplified) is that we use a 1 to represent giving power to a wire and 0 to giving no power (aka grounding) a wire. Now we have a way of talking about electric potential. Now we can string together these 1's and 0's into sequences and create a

special circuit to convert them into electric flow on the wires.

Now we have abstracted away the electrical engineering side of computers so that we don't have to worry about it. Now all we need to do is create the correct sequences of binary numbers to make the computer do stuff. Computers only understand this binary stuff. Well, lets give these special sequences of binary names. Names such as ADD (add two numbers together), MOV (move data around). We often refer to these names as instructions. We can now create a set of instructions so that we don't have to remember the binary sequences. Instead, we can create a Central Processing Unit (CPU) which has all the circuitry to execute these instructions.

Here is an example of an ARM (ARM is a type of instruction set that usually runs things like your phone and small devices) assembly program that I wrote in college:

```
.equ MAX,          1000
.equ SWI_PRINT_INT, 0x6B      ;Constants
.equ SWI_PRINT_CHAR, 0x00
.equ ASCII_SPACE,   0x20
.equ EXIT,          0x11

_start:

MOV r5, #1                ;loop counter
MOV r6, #1                ;5 counter

LoopCondition:
    CMP R5, #MAX           ;compare counter with 1000
    BLE LoopBody           ;choose address to load
    BGT LoopDone

LoopBody:
    CMP R6, #5             ;compare the 5 counter with 5
    BLT IfPrint
    BGE ElseReset

IfPrint:
    MOV r1, r5             ;move value for printing
    MOV r0, #1             ;swi stdout arg for printing
    SWI SWI_PRINT_INT      ;print integer to stdout

    MOV r1, #ASCII_SPACE   ;print a space between numbers
    SWI SWI_PRINT_CHAR     ;swi arg for printing a char
    B Increment

ElseReset:
    MOV R6, #0             ;reset 5 counter to 0

Increment:
    ADD R5, R5, #1
    ADD R6, R6, #1         ;increment counters
    B LoopCondition        ;restart loop

LoopDone:
    SWI EXIT               ;exit program

.end
```

Figure 1: ARM assembly program that prints all the integers greater than 0 and less than 1000 that are not divisible by 5

Please do not worry about what any of that stuff means. Not even I remember what everything does, and I was the one that wrote it. Yes, people used to write programs like this. In fact, the game Roller Coaster Tycoon was written in assembly.

Although better, this still looks terrible to remember and write. However, what if we could create a language that is much easier for humans to understand that can be translated into these instructions and thus be run by the computer? In comes high level programming languages such as C, C++ and yes, Java (java is a tad bit different, but the point is the same).

HELLO, If you were skipping over all that other stuff that I said you could, now is a good time to start paying attention. According to Wikipedia , "A computer program is a collection of instructions that performs a specific task when executed by a computer." As a programmer, you write these instructions. We use high level programming languages, such as Java, to abstract away the specifics of that internal workings of the computer, to write instructions that both humans and computers can understand. Reread that statement...HUMANS and computers can understand. This is absolutely critical, H-U-M-A-N-S and computers.

Now, we start with a plain text file such as the one shown in the video series

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

This would be saved into a plain text file such as, HelloWorld.java. This file is then compiled. This simply means that we use another program, which we refer to as a compiler, to translate our plain text code into machine instructions. So we are taking the English that is easy for humans to understand and translating it into machine instructions that the machine understands. This new file with the machine instructions is called a program. When you double click that program on your desktop, the machine instructions get loaded into memory and executed one at a time by the Central Processing Unit (CPU). Also a fun fact, your operating system, such as Windows 10, Ubuntu Linux, MacOS, Android, IOS, etc, is a program. So you as a programmer write the program, which gets compiled by another program and loaded/ran using your operating system, which is also a program.

1.2 The Structure of a Java program

This course will focus on programming using Java. Why use Java? Well, that will be covered in a later section. If we again look at the program from Lecture 1 and 2,

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

We can talk about a few things found here. In Java, everything is a class. If you do not know what a class is, don't worry, we cover that in the upcoming chapters and that phrase will make sense. Java is Object Oriented, again, this will make sense later. The thing I want to talk about here is "main". When you write your program, the compiler needs to know where the program starts. Most languages require some entry point into your program so that it knows where to start and most of the time we are required to call it "main". Meaning this is the main entry point of the program. Every program you write in Java will have this "main". It tells the compiler where to start and then just reads the code from top to bottom, left to right (Just like how you would read anything else in English).

2 Basic Syntax (Video Series Lecture 1 and 2)

In this section we will learn about Java keywords, what variables are what types are and how to print output to your console.

2.1 Keywords

If we again look at the code from the video series,

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

we will see certain things are highlighted colors by the editor. We call this syntax highlighting. It helps us distinguish between different things in our code, such as keywords. Please note that the colors and the syntax highlighting in general all depends on your editor, it has nothing to do with Java. Now from this code snippet, we can see that public, class, static and void are all highlighted blue. The editor does this because it knows that all of these words are Java keywords.

A Keyword is a word that is reserved by the programming language. This means that you cannot redefine what the word public, class, void, etc, means. Java has defined for you what they mean and it is your job to learn what they mean and how to use them. In a sense, the keywords are the Java language. Here is a list of the Java keywords as found in the Java documentation at, [<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html>](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html)

abstract	continue
assert***	default
boolean	do
break	double
byte	else
case	enum****
catch	extends
char	final
class	finally
const*	float
* not used	
** added in 1.2	
*** added in 1.4	
**** added in 5.0	
for	new
goto*	package
if	private
implements	protected
import	public
instanceof	return
int	short
interface	static
long	strictfp**
native	super
	switch
	synchronized
	this
	throw
	throws
	transient
	try
	void
	volatile
	while

Whenever you write these words in your plain text file, Java will use the definition that they have defined for these words. If you try to redefine them somehow, the compiler will probably give you an error because it simply isn't allowed.

3 Variables and Types (Video Series Lecture 3 and 4)

3.1 Java Types (Primitive Types and Strings)

Some of these keywords are called data types, particularly, primitive data types. This basically means that these keyword types are used to represent raw data, such as the binary representation of a number. In the video series, we created this table,

Name	Type	Example
int	Whole integers	25
double	Decimal numbers	13.765
char	Single characters	'h'
String	A string of characters	"Hello world"
boolean	true or false	false
short	Whole integers	(none)
long	Whole integers	(none)
float	Decimal numbers	(none)
byte	*	(none)

Figure 2: Table from video series lecture 3

Notice that all of the names in the table are in the reserved keywords list from 2.1, except "String". We include the String in the table because it is extremely common to use, however, it is not a primitive data type, it is an object. For now, there is no need to worry about what the difference actually is.

Take a look at the above table and notice that int, short and long all store whole integers. Similarly, float and double both store decimal values. So why have these keywords that all do the same thing? The answer is, they store different sizes. Sizes of what? Let's take an example, int. Int stores the binary representation of the whole integer. so to store larger numbers, we need more bits (more 1's and 0's) to uniquely represent that number. And thus we need more storage space to store those extra bits. Take a look at the following

table and notice the which types have more storage space and thus can store a larger range of values.

Type	Size	Range
boolean	1 bit	true or false
byte	8 bits (1 byte)	[-128, 128]
short	16 bits (2 bytes)	[-32768, 32767]
char	16 bits (2 bytes)	[0, 65535]
int	32 bits (4 bytes)	[-2147483648, 2147483648]
long	64 bits (8 bytes)	$[-2^{63}, 2^{63} - 1]$
float	32 bits (4 bytes)	32-bit IEEE 754 floating-point
double	64 bits (8 bytes)	64-bit IEEE 754 floating-point

Notice that long has a much larger range for the values it can handle in comparison to int due to the extra space that the keyword type allows. When your code is compiled, these types tell the compiler how much space to set aside for the value it is going to store that. But what happens if we go over the range? This is called overflow. For example, in the case of int, if we stored larger than 2,147,483,647, we would have integer wrap. This means that we have run out of unique ways of representing integers so we lose some of the bits. The resulting bits that get stored just so happens to be the same as some other number in that range. So if we go slightly over the max number in the int range, what gets stored some negative number on the other end of the int range.

A classic example of integer wrap is the Y2K bug. You can read about it [here](#). But essentially we were storing years as two digits 00 to 99. So the year 1995 would be stored as 95 and the year 1900 would be stored as 00. This was done because back in those days, we had to be extremely conservative with memory storage space. Well in the year 2000 (just like when we reach the end of the int range) we ran out of unique ways to store years. So the year 2000 would be stored as 00 which is actually the opposite end of the range, namely year 1900. The year wrapped around to the beginning of the range. So we thought the world was going to burn, planes would crash, the stock market would crash, aliens would invade, cows would be abducted. But nothing really happened. It is however an example of overflow, aka integer wrap in this case.

In case you were wondering what this IEEE 754 floating-point thing is. IEEE is a group that sets standards for companies who make computers. This way, computers can be universal and interact with each other despite being different hardware and software. One such standard is the way we store decimals in binary. You can read about it <https://en.wikipedia.org/wiki/IEEE_754> also if you would like to read more about primitive data types, check out the javadocs tutorials found at <<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>>

3.2 Variables

Variables are ways that you can store things under a name. From the video series, we show that variables take the form of

TYPE	<i>VARIABLE_NAME</i>	=	VALUE
------	----------------------	---	-------

Variables are often used to store values that you do not know until the program is running. Often times we need to pass around a value that is often changing. In the video series, we showed basic usage of declaring variables with a value.

```
public class HelloWorld {  
    public static void main(String[] args) {  
        int number = 5;  
        double score = 97.3;  
        char letter = 'h';  
        String hello = "Hello World!";  
        bool understands = true;  
    }  
}
```

We can also do things like copy values from one variable into another or reuse variables that are already declared.

```
public class HelloWorld {  
    public static void main(String[] args) {  
        int number = 5;  
        double score = 97.3;  
        char letter = 'h';  
        String hello = "Hello World!";  
        bool understands = true;  
  
        int newNumber = number;  
        number = 0;  
        understands = false  
    }  
}
```

In this case, newNumber will equal 5 and then on the next line, number will equal 0 but newNumber will still equal 5 since it made its own copy before number was reassigned.

4 Math in Code (Video Series Lecture 5)