Lecture 01: Introduction

SE115: Introduction to Programming I

Course Information

- Introduction to Programming I
- 2 hours of theory, 2 hours of lab
- This is the **fundamental** course of Computer and Software Engineering programs.
    - You will be developing programs during your education and in your professional life.
    - This is the groundwork for the courses you will take in the following years.
    - Invest in this course, and it will pay back immediately, starting the next semester:
        - SE116 - Introduction to Programming II assumes that you have **mastered** this course.

Course Information

- The course book is Deitel’s “Java How to Program”
- We also have an online book, “Java Early Objects”.
  - The online book have interactive applications which will help you learn programming.
  - We will provide a link on the Blackboard system in the upcoming days.
- These slides will be the main “lecture notes” - you should be taking notes in the classroom as well.
- Use the course book and the online book to study at home.
- The syllabus states that you have to study 5 hours out of the class every week.

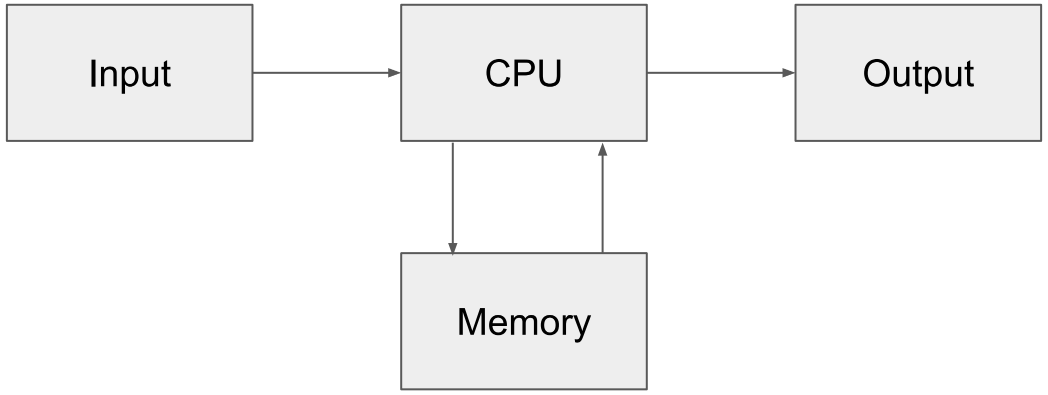
Questions?

Introduction to Computers

- The term “computer” has changed considerably in the last few decades.
- A computer is no longer a PC or a laptop anymore.
- Mobile phones, gaming consoles, and numerous other devices in “internet of things” are now capable of running software.
- Basically, a computer is made up of “hardware” and “software” parts.
- The hardware is useless without the software, and the software cannot run on its own, it needs the hardware.

Basics (Hardware)

- Computer architecture can be simplified as shown.
- The central processing unit (CPU) takes an input, operates on it while using memory, and produces an output.



Basics

- The CPU architecture can change, depending on the device.
- The most common ones are
  - the x86 architecture, in Intel and AMD processors, such as i5 and i7. This architecture is from the family of **c**omplex **i**nstruction **s**et **c**omputers (CISC). Most laptops and desktop computers use this architecture.
  - the ARM architecture, such as the latest Apple Silicon processors as well as most mobile processors. This architecture is from the family of **r**educed **i**nstruction **s**et **c**omputers (RISC). Since it is more energy efficient, it is used mostly on devices that operate on battery.

Basics

- The CPU contains registers for memory, as well as a cache when larger memory is required.
- A random access memory (RAM) can also be accessed.
- The input and the output are handled by their own devices. For example, the keyboard and the mouse are inputs, and the screen is an output.
- It is possible to use files for both inputs and outputs.
- These devices are usually connected via a “motherboard” or a “logic board”.
- Here is what it looks like:

A Motherboard

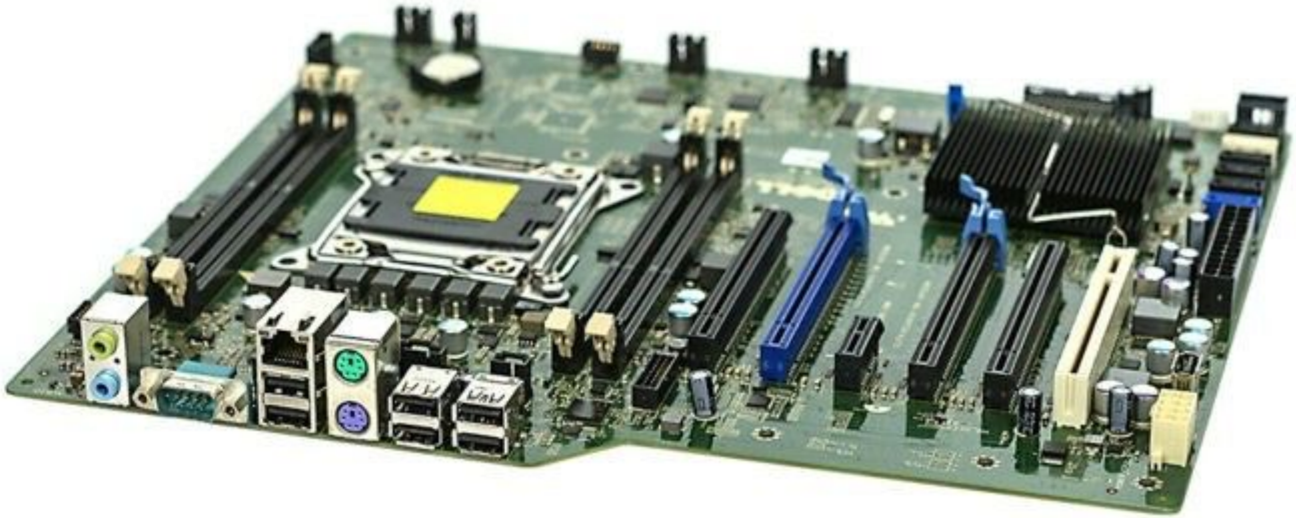


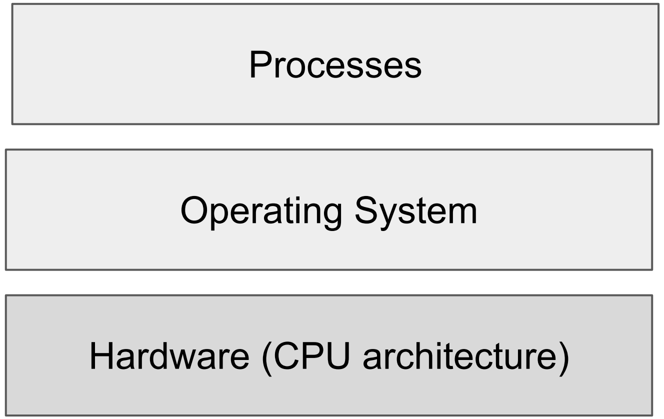
Image reference: https://wikiimg.tojsiabtv.com/wikipedia/commons/b/b7/Computer-motherboard.jpg

Basics (Software)

- The bare metal is useless without the software.
- However, it is not that simple to control the hardware - they require low-level instructions even for simple things.
- To manage the hardware, we use another software called the “operating system” (OS).
- The OS is important because it manages the resources, such as the CPU, the memory and the file system, and allows us to run processes on itself.

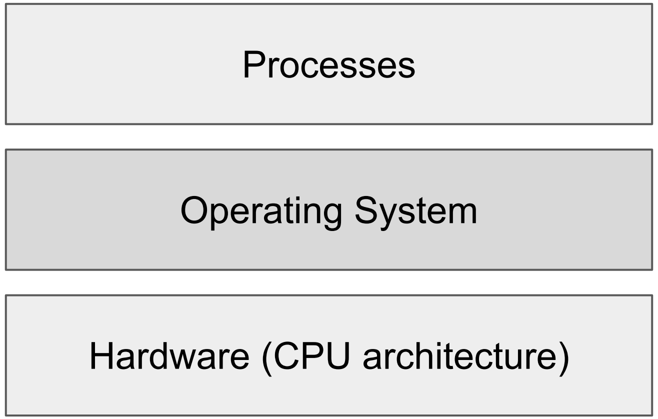
Basics (Software)

- The figure emphasizes the CPU architecture.
- That is because the architecture dictates the instructions we can run on the CPU. The instructions for a x86 cannot be executed on an ARM processor.



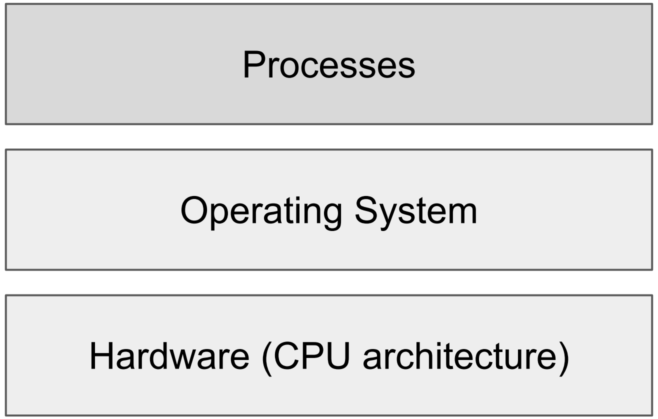
Basics (Software)

- The operating systems also have their own families.
  - Windows: a popular choice on the desktop and laptop computers.
  - Unix: a family of operating systems that date back to late 1960s. Unix has a lot of flavors, Linux and macOS being the most popular ones. While you might be exposed to Windows, Unix is the most common operating system in the world. The mobile devices, both Android and iOS are Unix-like. Most of the web servers also run on Unix-like operating systems.



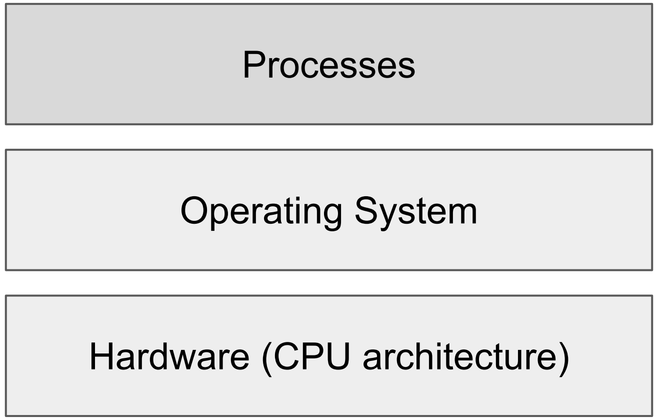
Basics (Software)

- When a program is running, we call it a “process.”
- **A process is a program in execution.**
- The files that we can execute are called “executables.”
- You can have several processes of the same executable; for example you can have a lot of Notepad instances running at the same time.



Basics (Software)

- The OS provides an environment for the program to run.
    - This is about providing access to CPU, memory, and I/O.
- The OS also provides a file system.



The File System

- Remember the computer architecture where there were input, output, the CPU and the memory.
- The memory is not persistent. All is lost when the process ends, or when the hardware is turned off.
- We need devices such as the hard drives to make them persistent.
- The file system provides a hierarchical structure that is basically composed of directories (folders in Windows) and files.
- We will get into details during the lab.
- Let’s focus on the files.

Text Files and Binary Files

- Text files are files that we can open with a “text editor”, such as Notepad or TextEdit, and read its contents.
- Then, there are “binary” files.
    - Here “binary” refers to the fundamental representation of data in computer science: 0 and 1.
- Images, videos, music files, and alike are all binary files.
- **A specialized program is required to view their contents.**
- Executable files are also binary.
- Before we can take a look at how they are represented, let me briefly discuss the fundamental representation of data in computer science.

Base 2 Numeral System

- Computers encode information using the base 2 numeral system.
- Why?
- The basic building block of modern CPUs is the transistor, which is used as a switch that can be either **on** or **off.**
- The **on** and **off** states are represented by 1 and 0 - also called “bits”.
- So what?
- We can use these two states to represent numbers:

Base 10	0	1	2	3	4	5	6
Base 2	0	1	10	11	100	101	110

Base 2 Numeral System

- Having numbers is enough to get things working.
- For example, if we say that each 8 of these values correspond to a value, then we can assign each value to a number, to a character, to an operator, etc.
- A **byte** is made up of 8 **bit**s.
- Since each bit can either be a 0 or a 1, there are 2^8 = 256 values a byte can store, starting from 0 to 255.
- When two bytes are used to represent something, then there can be 2^{16} = 65536 values, starting from 0 to 65535.
- For four bytes, it is 2^{32} = 4.294.967.296, from 0 to 4.294.967.295.

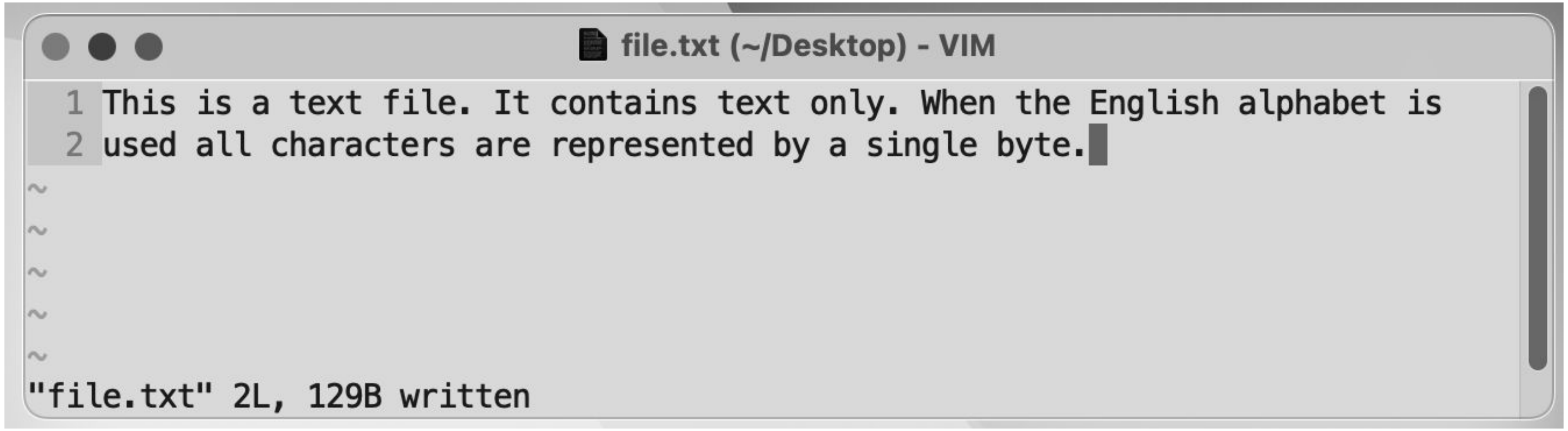
Base 2 Numeral System

- Anything we can do with base 10, we can do so with base 2.
- It is just a representation.
- As humans, we are fluent in base 10, but starting today, you will also be fluent in base 2.
- Now, let’s take a look at a text file up close.

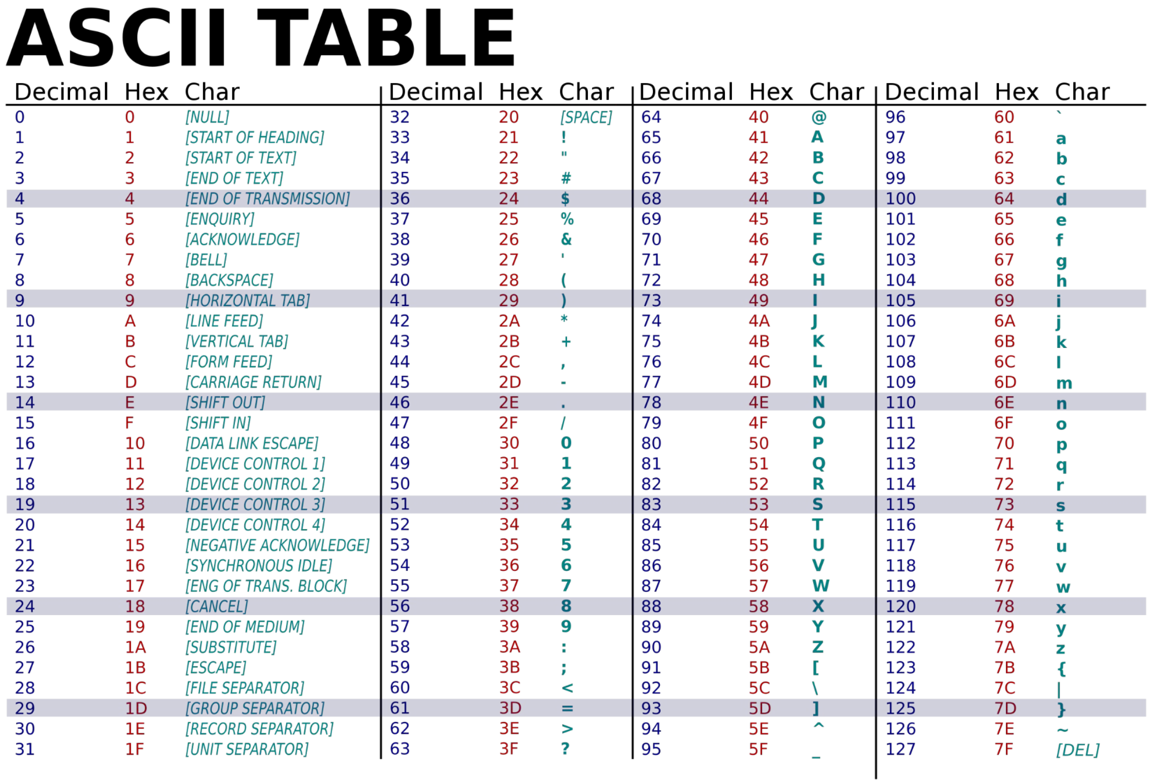
THERE ARE ONLY 10 TYPES OF PEOPLE.
THOSE WHO UNDERSTAND BINARY AND THOSE WHO DON'T

Text Files

- Notice the “129B written” - If you count the number of characters, including spaces, there are 129 of them. Each 1 byte long.
- So, let’s take a look at the ASCII table to check the corresponding integer values for each character.



```
This is a text file. It contains text only. When the English alphabet is
used all characters are represented by a single byte.
~
~
~
~
~
"file.txt" 2L, 129B written
```


ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

American Standard Code for Information Interchange

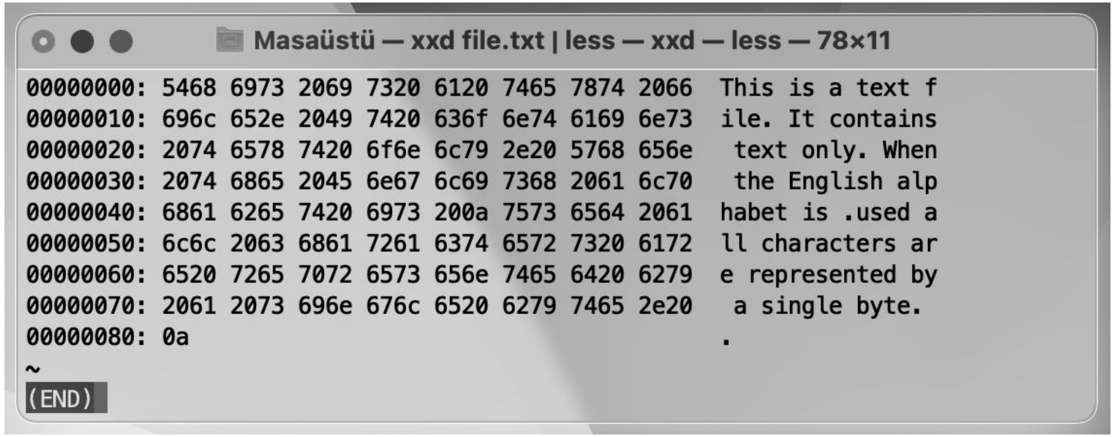
ASCII codes represent text in computers

Image from: https://commons.wikimedia.org/wiki/File:ASCII-Table-wide.svg

Binary, Decimal, Hexadecimal

- Hexadecimal is base 16… It uses A, B, C, D, E, and F for values 10, 11, 12, 13, 14, and 15, respectively.
- So, $(10)_{16}$ is 16. $(F0)_{16}$ is 15x16 = 240. $(FF)_{16}$ is (15x16) + (15x1) = 255.
- It is fun, isn’t it?
- The ASCII table also shows hexadecimal values even though they range from 0 to 127, which uses 7 bits, even less than a byte.
- Notice, however, there are no accented characters in this 127 characters.
- Let’s see how the previous text file is stored as “bytes”.
- To do so, we use a hex viewer. Since it is a hex viewer, it shows values in hexadecimal.

Revisiting the Text



Masaüstü — xxd file.txt | less — xxd — less — 78×11

```
00000000: 5468 6973 2069 7320 6120 7465 7874 2066  This is a text f
00000010: 696c 652e 2049 7420 636f 6e74 6169 6e73  ile. It contains
00000020: 2074 6578 7420 6f6e 6c79 2e20 5768 656e   text only. When
00000030: 2074 6865 2045 6e67 6c69 7368 2061 6c70   the English alp
00000040: 6861 6265 7420 6973 200a 7573 6564 2061  habet is .used a
00000050: 6c6c 2063 6861 7261 6374 6572 7320 6172  ll characters ar
00000060: 6520 7265 7072 6573 656e 7465 6420 6279  e represented by
00000070: 2061 2073 696e 676c 6520 6279 7465 2e20   a single byte.
00000080: 0a                                       .
~
(END)
```

$(54)_{16}$ = 84, T

$(68)_{16}$ = 104, h

$(69)_{16}$ = 105, i

$(73)_{16}$ = 115, s

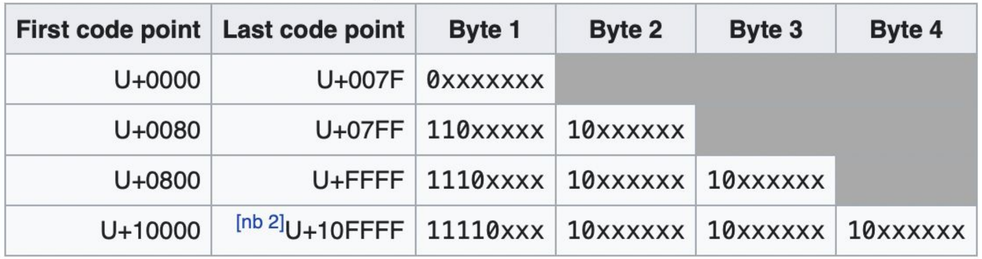
$(20)_{16}$ = 32, [SPACE]

… and so on…

Encoding on text files

- A final word: what about Turkish characters? Or characters that are not Latin-based? Arabic, Russian, Chinese, Japanese, etc…
- The UTF-8 encoding is developed with backwards compatibility to ASCII.
- You can read about it online, but basically they use the significant bits to encode whether a single or more bytes are used to represent a byte.
- If the byte starts with a 0, then a single byte is used. If it begins with 110, then two bytes are used, etc. up to four bytes, encoding up to 1,112,064 characters.

Code point ↔ UTF-8 conversion



First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0xxxxxxx			
U+0080	U+07FF	110xxxxx	10xxxxxx		
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
U+10000	[nb 2]U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Binary Files

- So, text files are easy.
- What about binary files?
- Anything that is not a text file is a binary file.
- Similar to how we need to know how text is encoded, we should know how the data is encoded for a binary file.

An Audio File

- For example, let’s consider a sound recording, which is uncompressed.
- A recording is done by taking a number of samples every second.
- Usually we take 44100 samples every second, a very precise recording.
- When there is no sound, then the value for that sample should be 0.
- When highest sound the sound device can capture is received, it should be… 65535 - this is because using two bytes is more precise than using a single byte.

An Audio File

- So, every two bytes is a sample, when we read 44100x2 of these bytes we should get one second of sound.
- Usually these files start with a “header” which includes information such as the number of samples, and number of channels (stereo? 5.1?) before real data begins.
- We need a program that can understand this content and play it so that the audio file can be used.

Executable Files

- Executable files are also binary files.
- They are different from an audio or an image file.
- These files will become processes. They will be loaded to the OS memory and will be executed.
- How is this possible?
- To run it, we must know what commands the CPU accepts and what their corresponding values must be, so that we can form a series of 0s and 1s that do something.
- OR…

Executable Files

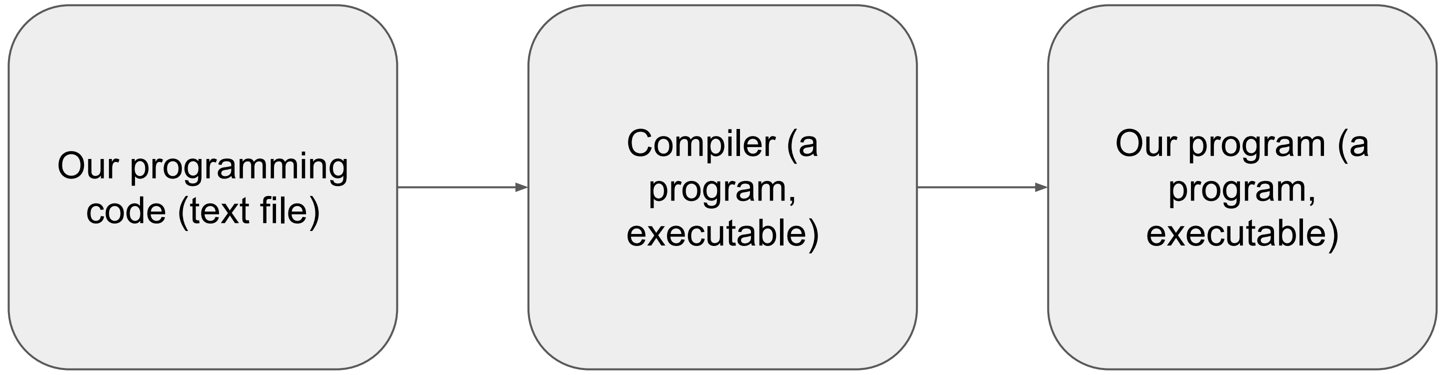
- Or we can use a “program” that can create another “program” from a set of instructions (code).
- The purpose of this program is to read the code and turn it into an executable file.
- We call these programs “compilers.”
- The code is written using a “programming language” as text files.
- So, we input text files to a compiler and it creates us a binary file that can be run on an operating system.
- In this course we will learn about “programming” and while we are learning it we will be using a “programming language”.

Programming, Programming Languages

- Programming is the act of creating a program by design and implementation.
- The design comes first; that is, you have to know what the program should do, step by step, so that you can write it in a programming language.
- Programming language is a formal definition of a “language” that has a syntax, a list of keywords, and other constructs.
- It is not like a regular language; in regular language you can form a sentence that is vague (not clear). In a programming language, there is no ambiguity or uncertainty.
- There has to be a set of rules because the instructions we write will be read by the compiler. The compiler must understand the instruction so that it can generate the machine code.

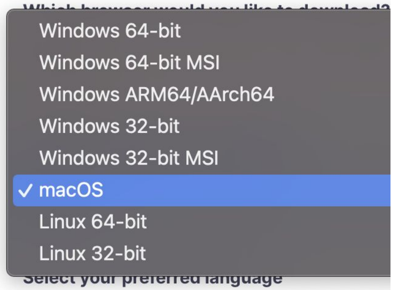
Programming

- We will be learning programming along with a programming language.


Our programming code (text file)
Compiler (a program, executable)
Our program (a program, executable)

Programs

- Programming languages such as C and C++ compile to executable files that can be run on the their target operating system and the CPU architecture.
    - For example, if you compile a C program on Windows and x86 architecture, then you can run it on a Windows machine that runs on an x86 CPU (i7), but not on Linux, even if the CPU is the same.
- Here is a screenshot from “Firefox” download page. Check out different versions of Firefox which has been compiled for different operating system and CPU architectures.

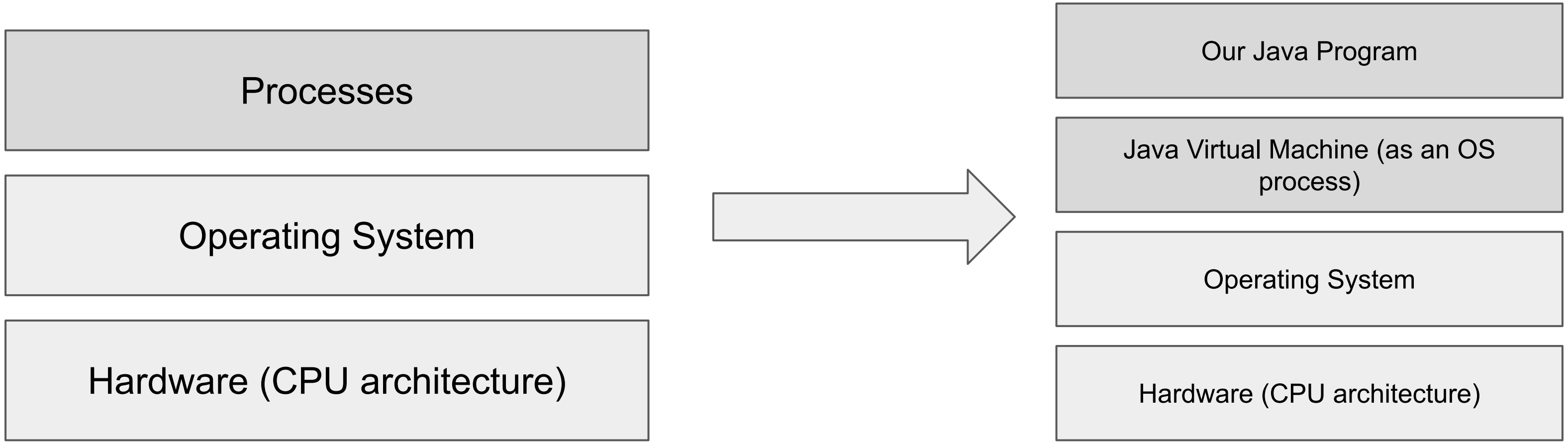


Java

- We will be using Java as our programming language.
- It is a “general use” “object-oriented” programming language.
- We will write Java code, compile it, and run it.
- However, in this regard, Java has come up with a unique approach.
- Instead of compiling for a specific OS and an architecture, Java code compiles to “bytecode” which can be run on a “Java Virtual Machine” (JVM).
- Java provides a JVM for each OS and architecture, and it is probably already installed on your computer.
- As long as there is a JVM, the bytecode runs.

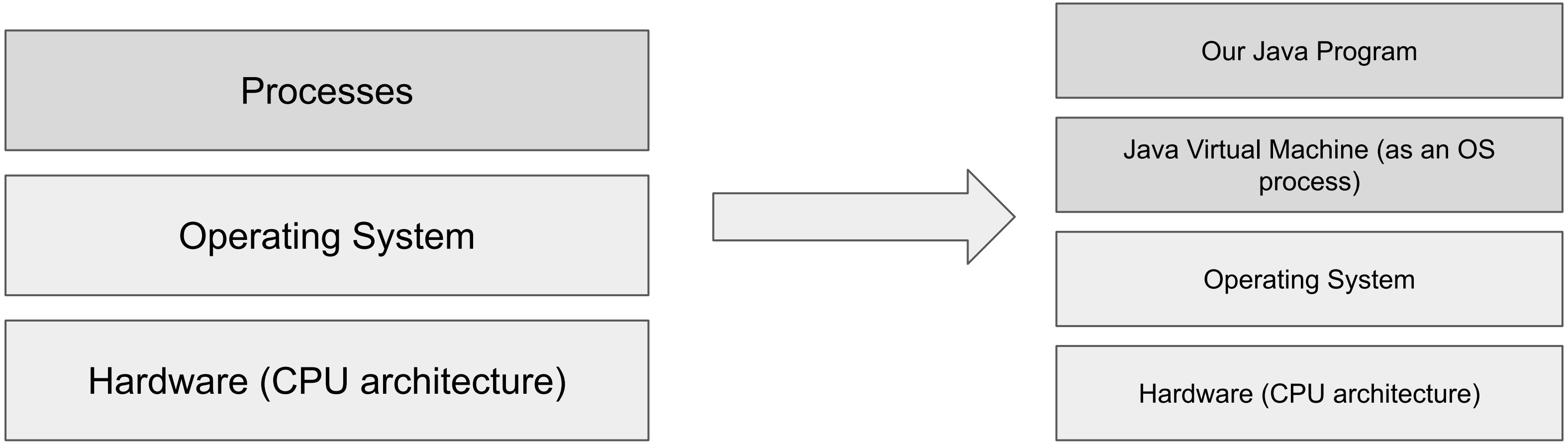
Java

- This frees us from the architecture & OS problem. You compile a Java program on Windows, and it will run on Linux as long as there is a JVM.
- Nothing is free, as you would expect.
- Instead of the program running on the OS, it now runs on JVM, and JVM runs on the OS. So, we add an additional layer…


Processes
Operating System
Hardware (CPU architecture)
Our Java Program
Java Virtual Machine (as an OS process)
Operating System
Hardware (CPU architecture)

Java

- Don’t worry! Computers are very fast these days. You won’t feel the difference. It will run almost as fast as a native process.
- JVM also has a few features which can translate your bytecode to the OS and arch below, such as Just-In-Time compiler, which compiles bytecodes to native machine code while your program is running.


Processes
Operating System
Hardware (CPU architecture)
Our Java Program
Java Virtual Machine (as an OS process)
Operating System
Hardware (CPU architecture)

Hello, world!

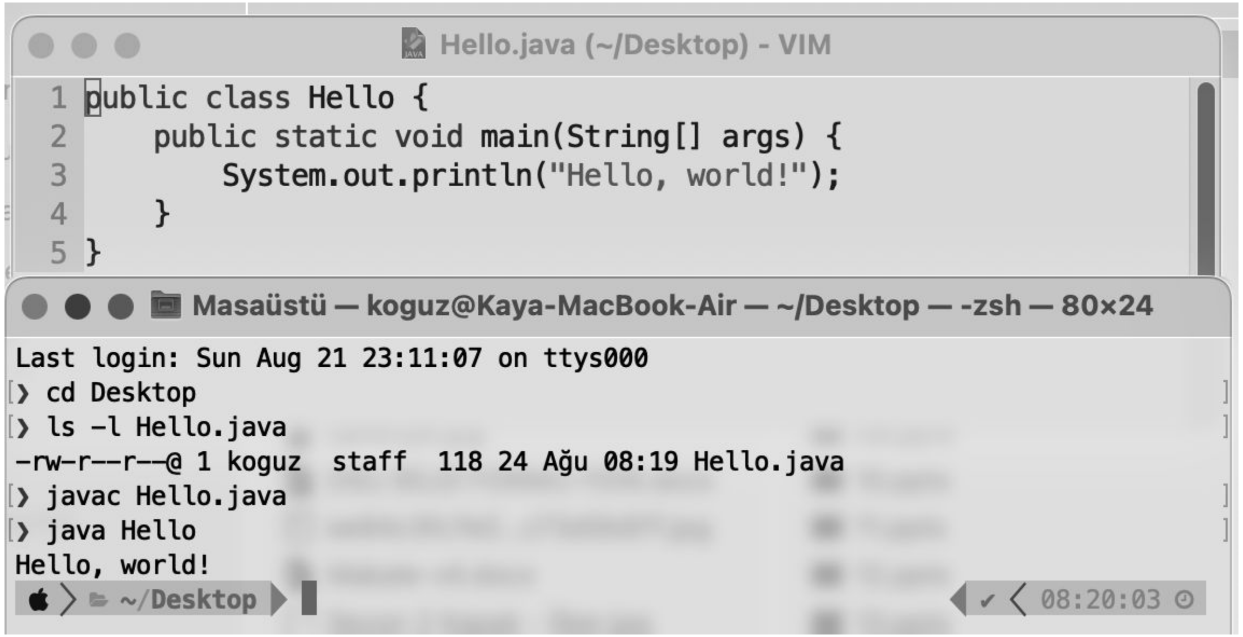
- Traditionally, we always write “hello, world” as our first program.
- You can write the following code using any text editor, but it has to be a text editor, not a word processor (such as Word).
- I will be using “Intelli J IDEA” as my “integrated development environment” (IDE).
- IDEs are very complicated programs. They have built-in features for project management, as well as debuggers, etc.
- For this course, all you need is a single text editor, and a compiler.
- You don’t need to create a “project” for each assignment.
- Most of the time, we will work on a single Java file, compile it, and then run it.

A Warning

- The IDEs have an advantage: they complete the code for you or provide a list of options to choose.
- Please remember that in quizzes and exams you will be writing code on a piece of paper: there is no IDE to help you complete your code for you.
- Therefore, until you are comfortable with writing without the help of another program, write the code using a simple editor: Notepad++ (Windows), vim (Linux and macOS).

Hello, world!

Here's the code I wrote on a beautiful summer morning. Let's talk about it.

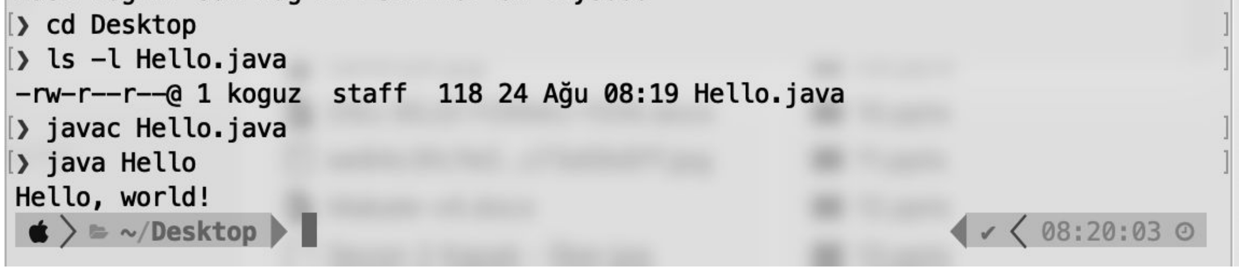

Hello.java (~/Desktop) - VIM

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

Masaüstü — koguz@Kaya-MacBook-Air — ~/Desktop — -zsh — 80×24

```
Last login: Sun Aug 21 23:11:07 on ttys000
❯ cd Desktop
❯ ls -l Hello.java
-rw-r--r--@ 1 koguz  staff  118 24 Ağu 08:19 Hello.java
❯ javac Hello.java
❯ java Hello
Hello, world!
 ❯ ~/Desktop ❯                                      ✓ ❮ 08:20:03
```

Hello, world!



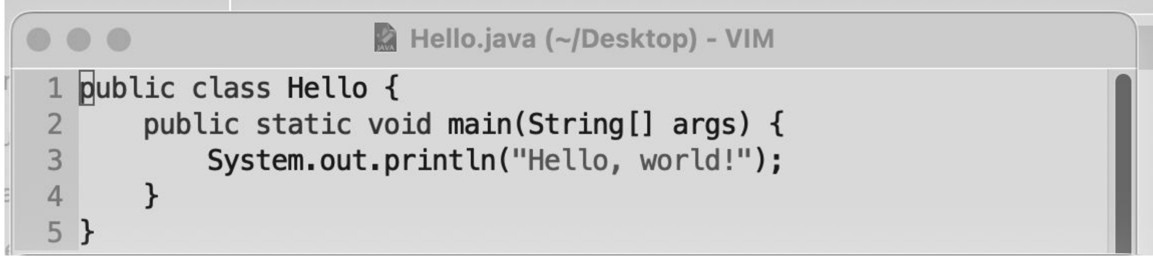
```
❯ cd Desktop
❯ ls -l Hello.java
-rw-r--r--@ 1 koguz  staff  118 24 Ağu 08:19 Hello.java
❯ javac Hello.java
❯ java Hello
Hello, world!
 ~/Desktop                                          ✔  08:20:03
```

- I wrote the code on a text editor (vim) and then saved it to my desktop.
- Then, I fired up a terminal, where I can run codes by typing them.
- First, I changed my directory to Desktop (cd Desktop).
- Then I have listed the file, to check if it is there (ls).
- Then I have run the Java compiler (javac) and provided the name of the file.
- It run without any warnings or errors.
- Then I run “java Hello” - the “java” command is required to run Java programs. They need the JVM, so this creates a JVM, and then my code is executed.
- On an IDE, the text editing, compiling and running takes place within the IDE, that is why it is called “integrated development environment”.

Hello, world!

- As your programs get more complicated you will be using several Java files, along with other files, such as text files, database connections, images, etc.
- Then you will need to learn about project management, too.
- For now, focus on the basics of the source code.
- Keep it simple.
- Let’s take a look at the source code.

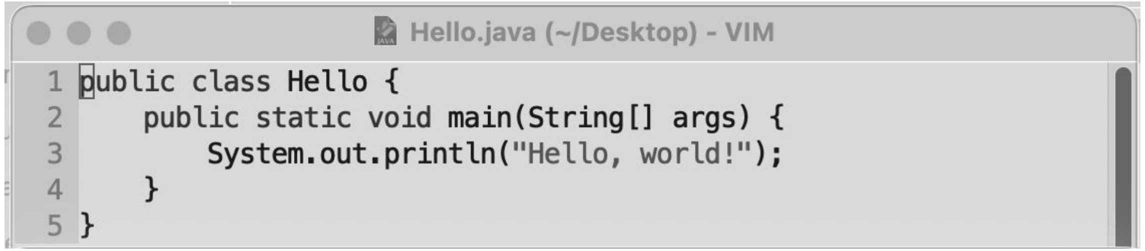
Hello, world!


Hello.java (~/Desktop) - VIM

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

- I know there are many keywords here.
- I will tell you what they are, but some of these topics are for SE116, the course where you will learn about the object-oriented programming.
- However, since we will be using Java, you are exposed to some of them.
- We will discuss objects after the midterm.
- Since Java is object oriented, it works with objects.
- An object is an instance of a “class” -
  - Class is the blueprint, for example, for cats. When I say “cat” you know what a cat is. However, think about the general properties of cats, not a specific cat. If you own a cat, let’s call him Topak, then it is an object that has all the properties of a cat.
  - So, class is the definition, object is the real thing.

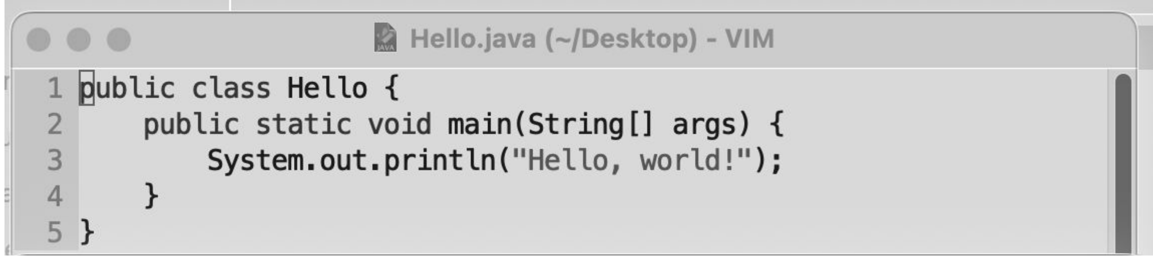
Hello, world!


Hello.java (~/Desktop) - VIM

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

- So, the first line includes two keywords: public and class.
- The keyword **class** says that we are defining a new class, in this case, that class is called “Hello”.
- The keyword **public** says that this class is public. That is, everyone can use it, it is not **private.** These are called “access modifiers” - they control who can access these classes. In this case, everyone can access Hello class.
- The curly braces are very important. The body of the class definition must be within these braces.

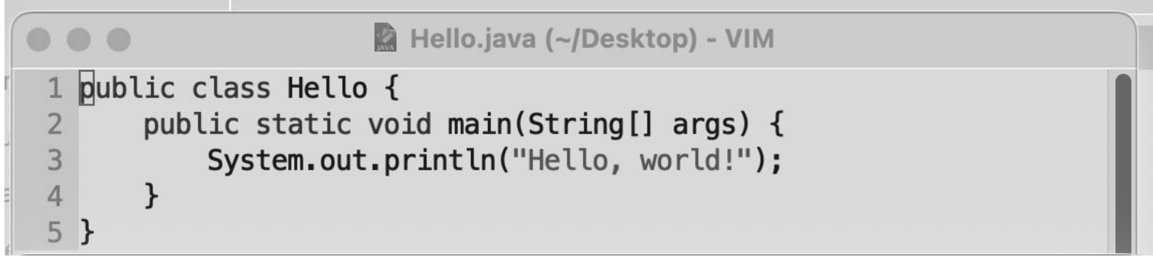
Hello, world!



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

- This class does not have much, it only has a function (method in OO).
- This is a very special function, it is called **main**.
- All programs require an entry point. For Java, (and also for C, C++, and C#) this is the main function.
- However, there are so many keywords before it.
- We know what public is, this method can be accessed (or called) by anyone.
- Who are these people calling these methods?
- In this case, it is us, but indirectly.
    - We called it when we ran “java Hello” - the JVM is created and it looked for the “main” and called it so that it can be executed.

Hello, world!

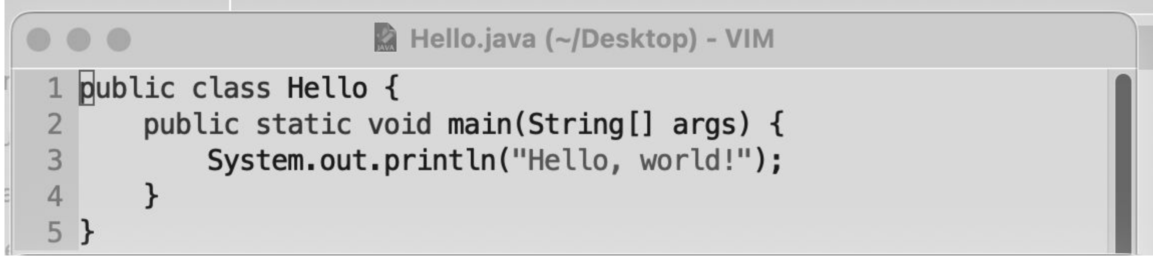


Hello.java (~/Desktop) - VIM

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

- The keyword **void** means that this function does not “return” anything. Void means “empty” - so the method does some stuff, but does not return a value.
- There are functions that return values or objects.
- The keyword **static** is tricky. To access any of the methods of a **class**, we need to have an **object** for that class.
- Right now, there are no cats (objects) around, so I cannot hear a meow sound. I need a cat object so that the method “meow” can be called.
- However, I can talk about cats (as a class). For example, I can say that there are 700 million cats worldwide (I looked it up, there are that many!).

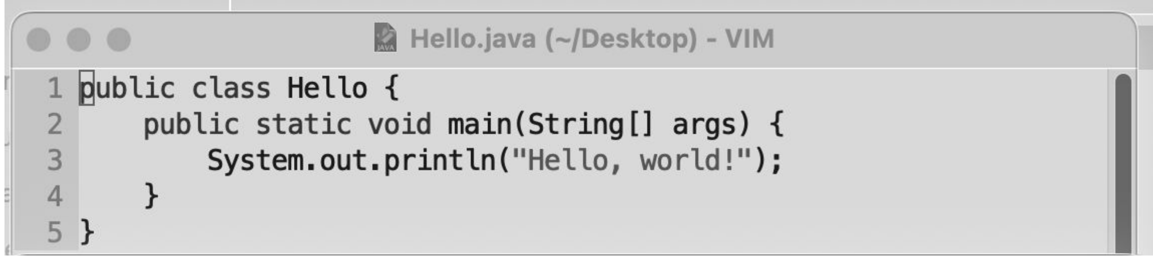
Hello, world!



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

- So, some properties may not require an object of that class. For those, we use the **static** keyword.
- In this case, it makes sense: How can there be objects just as the program is about to start? So, the main function must be static.
- The (String[] args) are the parameters of the function.
- This means that main takes a list of text, called args.
- Remember “java Hello” and “ls Hello.java” commands from the screenshots.
- In both cases we have passed arguments to these programs (java and ls).
- If we want to pass an argument to “Hello” program, then we do so by “java Hello arg1 arg2” and these “arg1” and “arg2” values are stored in the (String[] args) list.

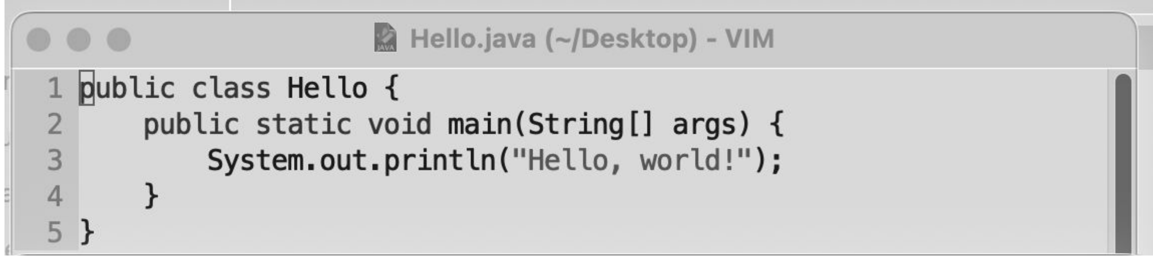
Hello, world!


Hello.java (~/Desktop) - VIM

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

- Not much left. Just a single line.
- Since there are no objects, we can say that the method **println** (print line) is static, and it takes a single parameter, the argument “Hello, world!”.
- The println method prints a line to the standard output (the text on the command line).
- System is a class, and it contains a static object called **out** from the class **PrintStream**.
- The println method is a member of the PrintStream class.
- Don’t worry about these classes and objects for now. We need them to print on the screen, that is all.

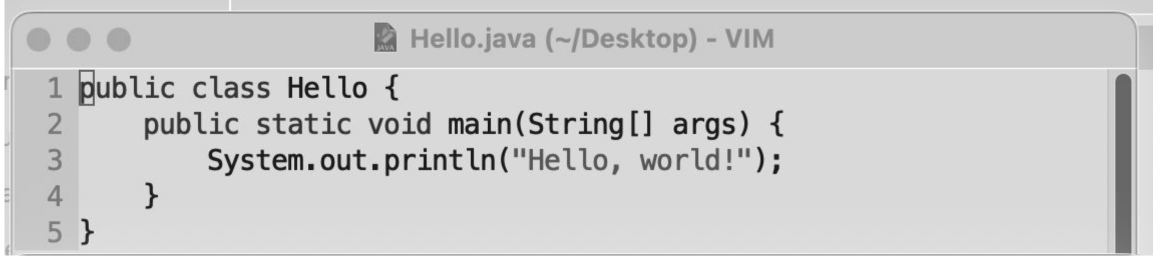
Hello, world!



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

- Notice that the main function is also surrounded by curly braces.
- The body of the method is between { and }.
- Once the closing brace is reached, the method ends, returning nothing (void).
- Also notice that the line that contains println ends with a semicolon (;).
- Such lines are called statements.

Hello, world!


Hello.java (~/Desktop) - VIM

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

- So, it is your turn!
- Type this code down.
- You can change the text between the double quotes “” - instead of hello world, you can print your name.
- Repeat the line.

That is all!

- That is all for this week.
- Next week we are shifting gears, so make sure you can run Hello World smoothly until then.