Java Fundamentals Course

From Hardware to Software

Computers: how do they work?

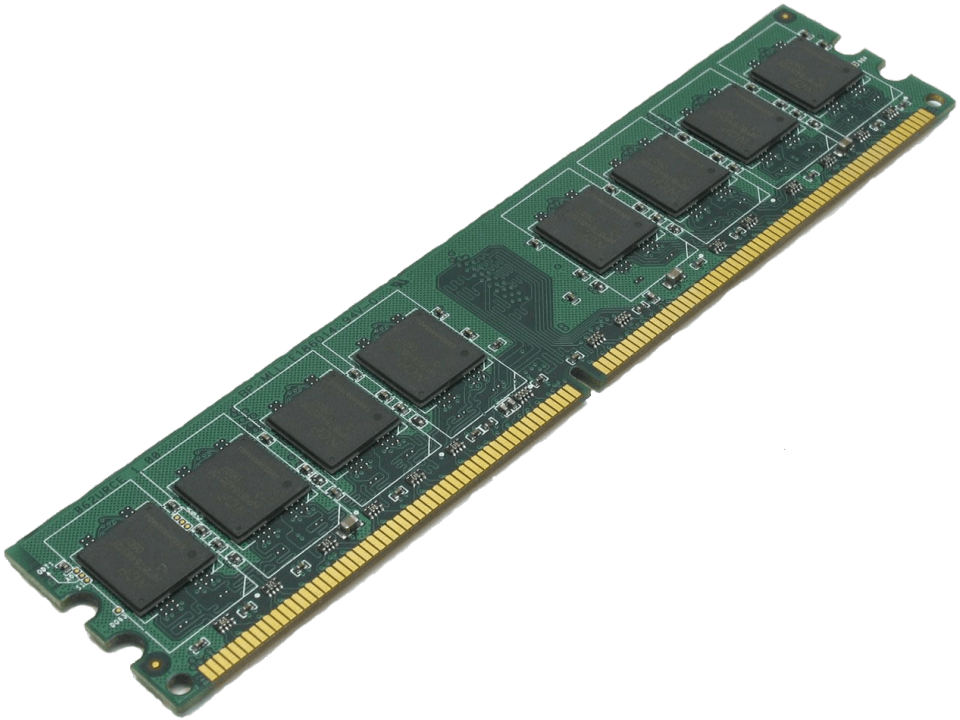
Computer components

- Memory

- Storage
- CPU

So how does a computer even understand this? For that we will need to dig into how a computer works. The internals of a computer are extremely complex, but we can simplify it to these three main components. There are many more components of course, but most of those are just to make these components work together, or to provide input (keyboard/mouse), or output (monitor).

Memory (RAM)



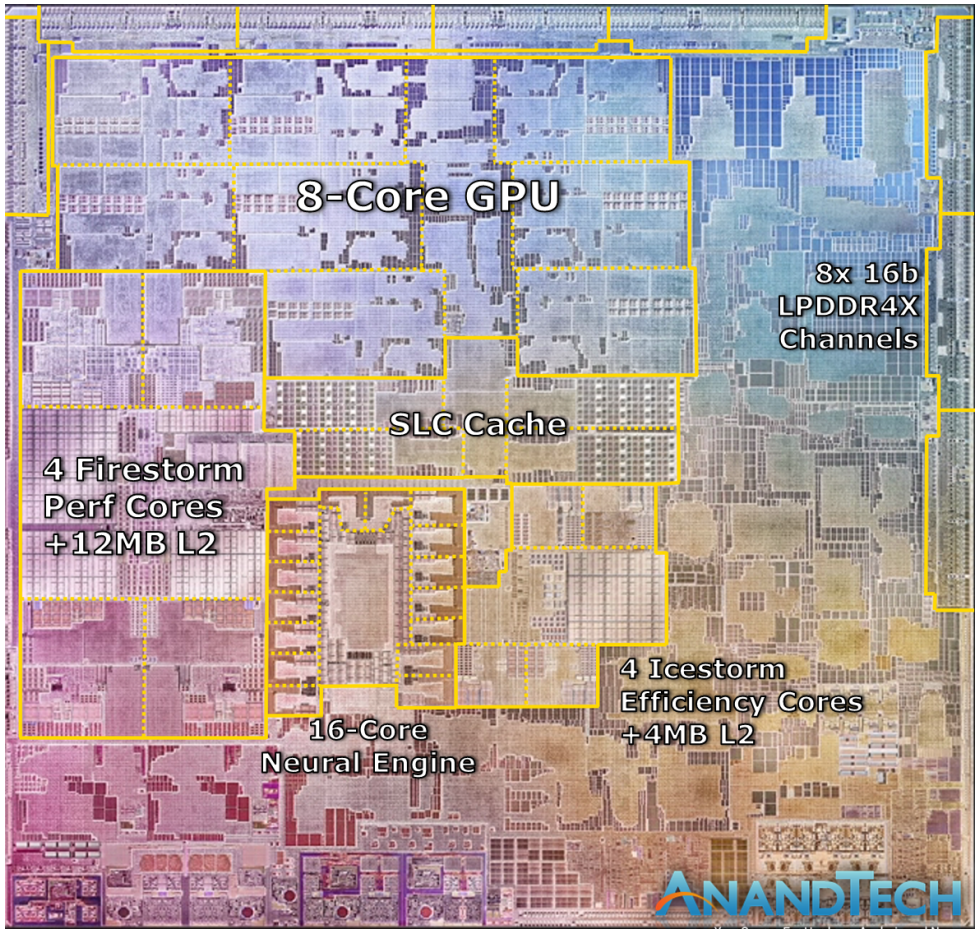
This is where all data is stored that you are currently working on. If you opened any program on your computer, the memory is where all that

program’s 'stuff' is kept. It is very fast but not permanent. If the power goes out, the data in memory is lost.

Storage

Because it would be nice to still have your files after you turn your computer off, we want to store them somewhere safely. This is what storage is used for.

CPU (Central Processing Unit)



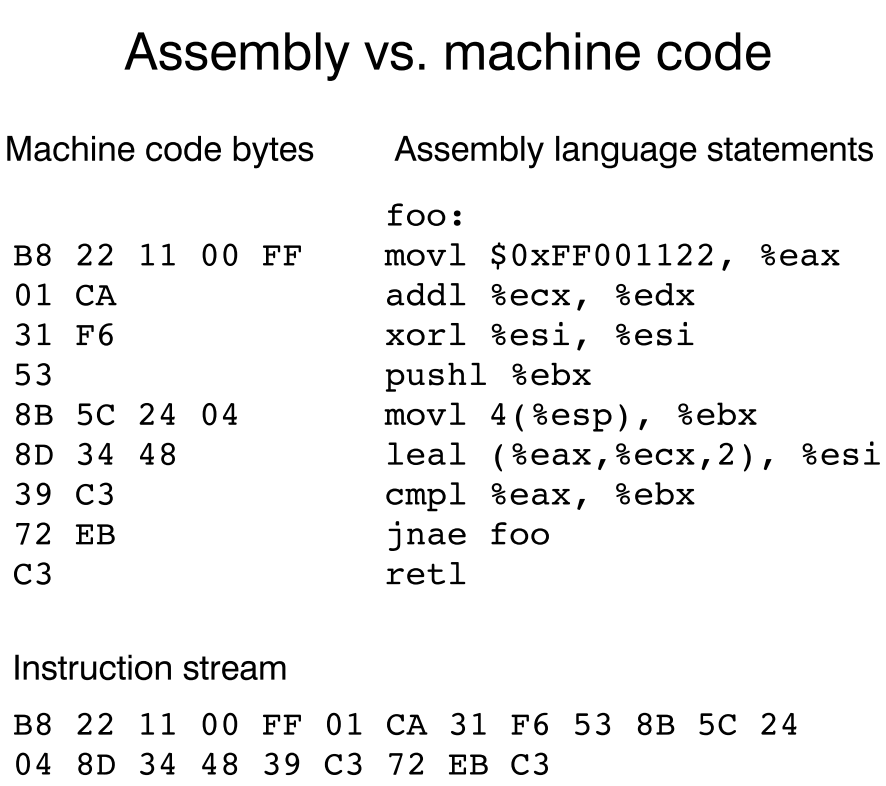
A CPU does all the work. It can perform basic operations like retrieving some small amounts of data from memory, saving it to memory, add numbers together, multiply etc. It doesn't do larger tasks though, for instance a CPU has no idea how to send an email or to open a website.

Software for Hardware

- CPU doesn’t "know" how to send an email or what "email" is
- Computers need instructions

So how does that work then? If a CPU doesn’t know how to send an email, how can we still instruct it to? Let’s start somewhere smaller.

Software for Hardware



Software for Hardware

- Assembly is very powerful, but is too detailed for high-level applications
- Most modern programming languages are high-level

High-Level Programming Languages

- Provides abstractions over computer specifics
- Uses natural language semantics
- Automates (or hides) a lot of computing operations
- Code is easier to comprehend and simpler to write

Abstraction

Abstraction is knowing **what** something does, but not **how**.

Abstraction example

- What you need to know about driving a car
    - Wheel
    - Pedals
    - Gear stick
- What you **don’t** need to know about driving a car

- Engine
- Transmission
- Electronics
- Steering system etc.

It’s like driving a car. You need to know what the car can do, and how to operate it. But you don’t need to know how an engine works, or the transmission. That is abstracted away from you. You just need to use the wheel, pedals, and stick.

Levels of Abstraction

- Driver
- Mechanic
- Engine manufacturer
- Scientists

These abstractions go further than that, there are always layers of abstraction. While you don’t need to know how an engine works, a mechanic does need to know to some extent. But they only know how to repair, replace etc, not how and why every bolt and piece is placed within an engine. The manufacturer does know that. And beyond that there will be some scientists that figured out the best efficiency for the engine based on thermodynamics etc. And perhaps that scientist doesn’t even know how to drive!

Abstractions in programming

- Software User
- Programmer

- Programming Language developer
- CPU manufacturer

And the same abstractions exist for Programming. You need to understand how to work with a programming language but you don’t need to know how a language works internally, or even how a CPU works. Just like how a user of the software you make doesn’t need to know how you’ve created the software.

Abstractions in programming

You don’t *have* to know how computers work on low level, but could it help?

Absolutely

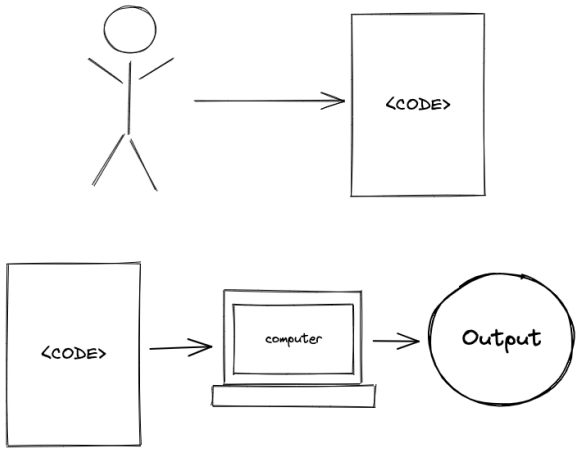
Absolutely it can help! Just like knowing how an engine works while driving a car, it can help with understanding issues and perhaps even prevent them. Actually lets go a bit deeper into one aspect of a computer right now.

What is Programming?

Definition

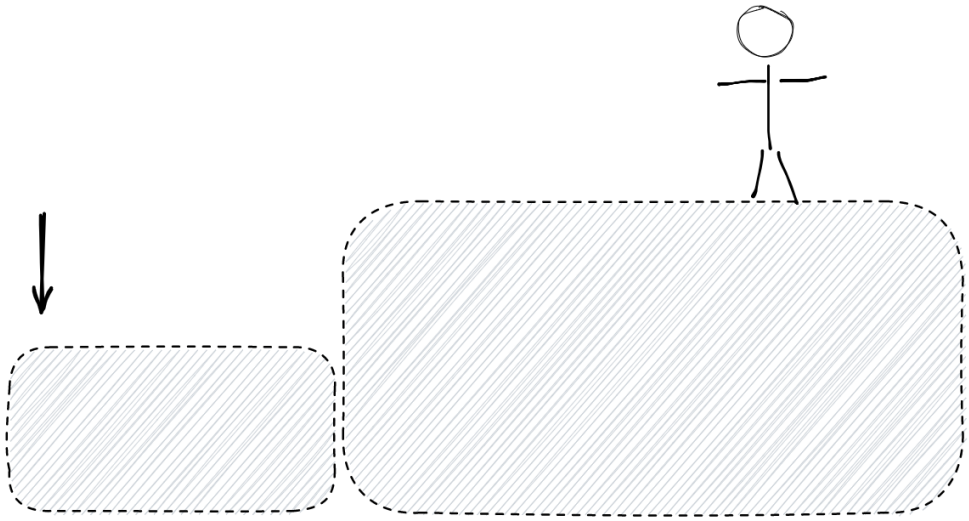
A Program is a set of instructions that a computer follows.

A Programmer is someone who creates that program.



This means there are at least two phases to programming: Writing the instructions and executing the instructions

Giving instructions



- Let’s help stickman get to the arrow

Let’s say we want stickman here to get to the arrow. Stickman doesn’t think for itself, you have to give very clear instructions to get there! [Ask for instructions]

The instructions

```
1: Turn and face the cliff
2: Walk towards the cliff until the edge
4: Jump off the cliff
5: Walk towards the arrow until touching the arrow
```

Detailed instructions

But what does this 'mean' for stickman?

```
2: Walk towards the cliff until the edge
```

You could see this one instruction as a bunch of actions that stickman must do:

```
1: Left foot forwards
2: Right foot forwards
3: If at edge, stop
4: Repeat from 1
```

This idea is called abstraction, and is something that will come back more often this course.

Bits and Bytes

Data

- What is data?

Data = information

Who can explain what data is?

Data is simply information. In computing, it is information that is stored in a way that is easily moved and processed.

Storing information

Vacuum tubes were one of the first ways to store information

In the earliest days of electronic computers around 1940, the only way to "save" anything, was with big shining vacuum tubes. They found a way to control which vacuum tube to turn on, and which to turn off through switches. This was the first way of saving data to a computer. This even made it possible to answer some simple questions

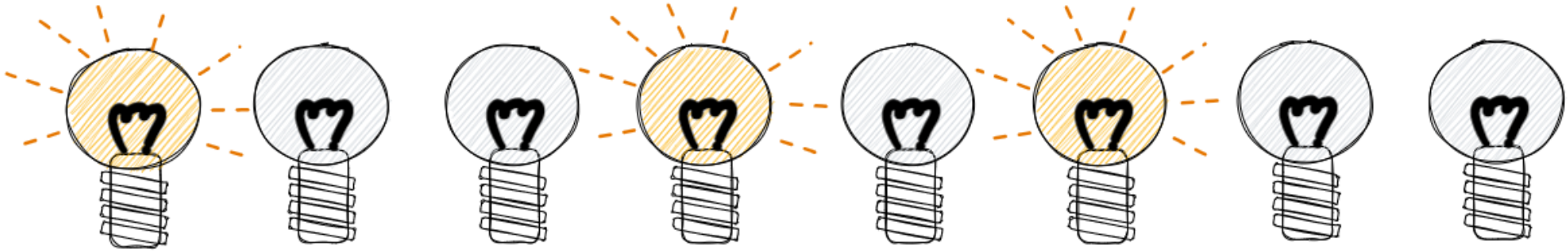
Vacuum tubes

Vacuum tube states:

- On / Yes / 1 / Blue
- Off / No / 0 / Red

So a vacuum tube has 2 different states, and each state we can interpret however we want. But we cannot answer more complex questions like this. If the question would be, "at what time will it rain tomorrow?" then we can't give a precise answer.

Vacuum tubes



Luckily we didn't give up there. Some smart people found out that if you group multiple of these vacuum tubes together, then you can have much more states! For instance here we have eight vacuum tubes, we can have 256 different states, which we each can interpret differently, giving us 256 possible answers to any question! As long as we agree on how to interpret these vacuum tubes. Now for the question "at what time will it rain tomorrow?" we can finally give a reasonable answer.

No more vacuum tubes

0 1 1 1 0 1 0 0

¬ As a number: 116

¬ ASCII: t

(https://www.asciitable.com)

Now vacuum tubes are of course a thing in the past, now we have much more efficient ways of storing memory. But the principle stays exactly the same. We have something in our computer that can either be on or off. And we can group those things together to create many different states. Generally speaking we don't talk about vacuum tubes anymore but about ones and zeroes. Where a vacuum tube is on, we say 1. If its off we say 0. One vacuum tube or "thing" that can be on or off is called a bit. 8 of them grouped together is called a byte. Why 8? Well, historical reasons. Mainly because if we group 8 together we can make letters! There are some common agreements made beforehand, one popular agreement called ASCII. If we would have the byte above, ASCII would say that this represents the lower-case letter T.

Storing larger data

So how would we represent `300`?

Or a full word like `hello`?

So a byte can store 256 different states, enough for one character, or, well, a number as high as 255 (including 0). What if we want to do even more? Add bytes together! Modern CPUs can handle 64-bits as one unit, that is 8 bytes together

64bit and 32bit

As (decimal) numbers:

64bit max value is `9,223,372,036,854,775,807`

32bit max value is `2,147,483,647`

Somewhere in the last decade computers have made the switch from 32 bit to 64 bit, being able to handle much higher numbers as a single unit.

Nice to know, but is it relevant to programming?

Yes.

A lot of programming languages still need you to make the distinction between 32bit numbers (or integers), and 64bit numbers. An 'integer' in Java is actually 32 bits, which is important to know when working with large numbers!

Programming languages

How does a programmer interact with a computer

What is a programming language

The tool a programmer uses to give instructions to the computer.

It converts your instructions into Machine Code

It’s a medium, a way of telling the computer what to do. The language itself is not something that a computer understands, there are often many

different steps between the code you write and the actual instructions that a CPU receives. And again this is for good reason: Abstraction!

Different languages

- Java
- Python
- C
- C++
- JavaScript
- PHP
- Ruby

…and many many more

The code example I gave you earlier was actually written in Java. But Java isn't the first language that has been created, and certainly isn't the last one. Actually, there are quite a few programming languages out there:

Why different languages

So why are there so many languages, if a language is just about giving instructions to a computer, isn't having one language enough?

Differences

- Ease of learning. Python is known to be easy to learn, but is not exceptionally efficient.

- Performance. C is known to be blazing fast, but harder to learn and code. Taking more time to program.
- Where they can be executed. Browsers can generally only run JavaScript, other languages may not even be an option

There are actually quite some differences between these languages in many different things: and there are many other considerations as well.

Is there a 'best' language?

For a company the "best" programming language depends on their use-case. If they need software that runs exceptionally fast, then you will likely see C or C++. If a company needs only a few scripts, one-off programs, that only need to run once in a while, then Python is a good choice. If the company is building an application to be run in the browser, perhaps a website, then you will likely see JavaScript.

What about Java?

- 2nd most popular programming language (as of december 2021, after Python)
- Relatively simple to understand, learn, and program with.
- Quite fast. Depending on how you measure, C is about 2x as fast, but Python is about 30x as slow.
- Platform independent, we will go into this later but it can be run just about anywhere because of the JVM
- Secure, that JVM also makes it secure.

So I’ve mentioned all these languages, but not so much about Java. Where does it lie between all other options?

These are some of characteristics of the Java programming language:

All of these points are very important for most companies. Java enables developers to write efficient code quickly, and being able to iterate over that code quickly as well so it can continuously be improved.

Syntax compared: Java

```
import java.util.Scanner;

public class Adder {
    public static void main(final String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter two numbers, separated by a space...");

        String input = scanner.nextLine();
        String[] numbers = input.split(" ");

        int numberOne = Integer.parseInt(numbers[0]);
        int numberTwo = Integer.parseInt(numbers[1]);

        int result = numberOne + numberTwo;

        System.out.println("The result is: " + result);
    }
}
```

Syntax compared: C

```
#include <stdio.h>
int main()
{
   int num1, num2, sum;
   printf("Enter first number: ");
   scanf("%d", &num1);
   printf("Enter second number: ");
   scanf("%d", &num2);
```

```
    sum = num1 + num2;
    printf("Sum of the entered numbers: %d", sum);
    return 0;
}
```

Syntax compared: Python

```
x = input("Type a number: ")
y = input("Type another number: ")

sum = int(x) + int(y)

print("The sum is: ", sum)
```

Note that while the code may be shorter for Python, this doesn't necessarily mean that it executes faster! Python actually runs the slowest of these three.

Syntax compared: Assembly

```
.LC0:
        .string "Enter first number: "
.LC1:
        .string "%d"
.LC2:
        .string "Enter second number: "
.LC3:
        .string "Sum of the entered numbers: %d"
main:
        push    rbp
        mov     rbp, rsp
```

```
        sub     rsp, 16
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        lea     rax, [rbp-8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC1
        mov     eax, 0
        call    __isoc99_scanf
        mov     edi, OFFSET FLAT:.LC2
        mov     eax, 0
        call    printf
        lea     rax, [rbp-12]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC1
        mov     eax, 0
        call    __isoc99_scanf
        mov     edx, DWORD PTR [rbp-8]
        mov     eax, DWORD PTR [rbp-12]
        add     eax, edx
        mov     DWORD PTR [rbp-4], eax
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC3
        mov     eax, 0
        call    printf
        mov     eax, 0
        leave
        ret
```

Executing code

```
import java.util.Scanner;
```

```
public class Adder {
    public static void main(final String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter two numbers, separated by a space...");

        String input = scanner.nextLine();
        String[] numbers = input.split(" ");

        int numberOne = Integer.parseInt(numbers[0]);
        int numberTwo = Integer.parseInt(numbers[1]);

        int result = numberOne + numberTwo;

        System.out.println("The result is: " + result);
    }
}
```

execute code Let’s go back to this example code. Let’s not focus on all the code right now, instead lets try to run it.

Output of the Adder

```
Enter two numbers, separated by a space...
1 2
The result is: 3
```

```
Enter two numbers, separated by a space...
9999999 2
The result is: 10000001
```

It asks for two numbers and then returns the value almost right away. It even works for larger numbers! And we can even run the same instructions on

a different computer at the same time.

That’s the benefit of computers, they are much faster than humans, they are very precise, and they will follow the instructions exactly.

Bigger numbers

What happens when we try an even bigger number?

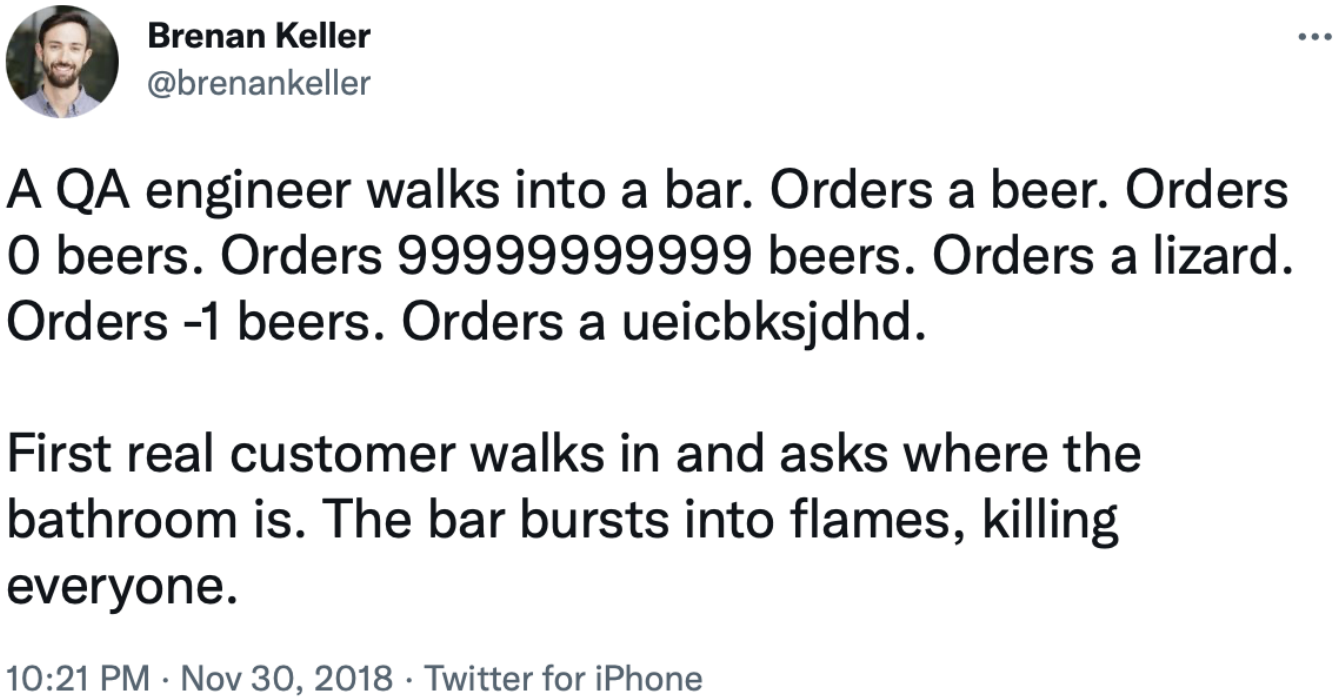
Talking about integers and such makes me think, what happens when we try an even bigger number? If we look at the code we see the words `Integer` and `int` a few times. And we just said an integer in Java only holds just over 2 billion. So what if we add 3 billion to 2?

Errors

```
Enter two numbers, separated by a space...
3000000000 2
Exception in thread "main" java.lang.NumberFormatException: For input string: "3000000000"
    at [...]
```

Well the computer just runs the program, it doesn’t think for itself. If it encounters something that the program doesn’t account for, for instance when it expects two numbers but gets only one, or a number that is too big, unexpected things can happen, often a big error occurs and the program stops. And this is only one case, what if we enter three numbers? no numbers at all? numbers with decimals? What if we enter words instead of numbers?

A developer walks into a bar…



A Computer isn’t dumb

…it’s precise.

A computer does exactly what it’s being told and nothing more. No assumptions. As a programmer you will need to be thorough in your instructions to it and think about the possible scenarios.

Programming = thinking ahead

That is where the programmer comes in. They need to think about all scenarios that can happen before they happen, because you won’t be able to fix them once the program is already being executed.

Exercise

Breaking down problems - Your morning routine/cooking

failure modes think about making a sandwich for instance? what if you’re out of bread?

Java Introduction

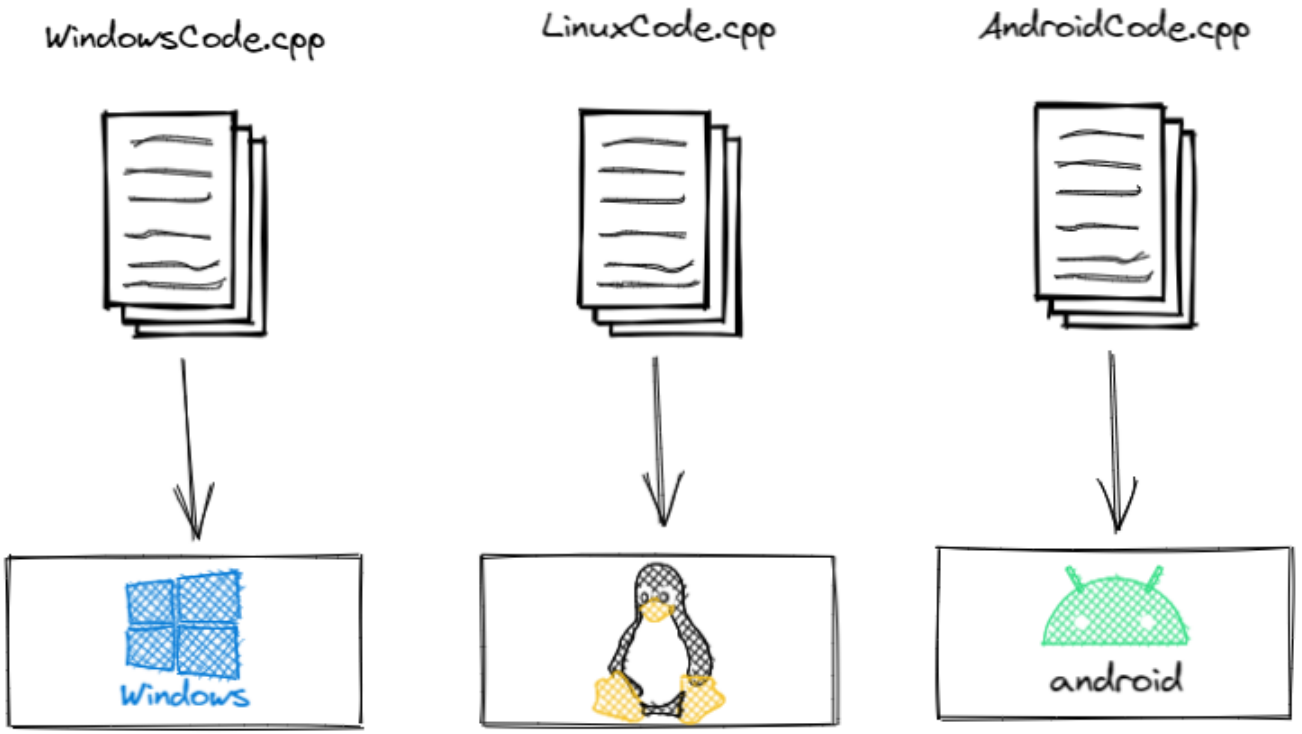
- What is Java?
- Why Java?

Java’s Origin Story

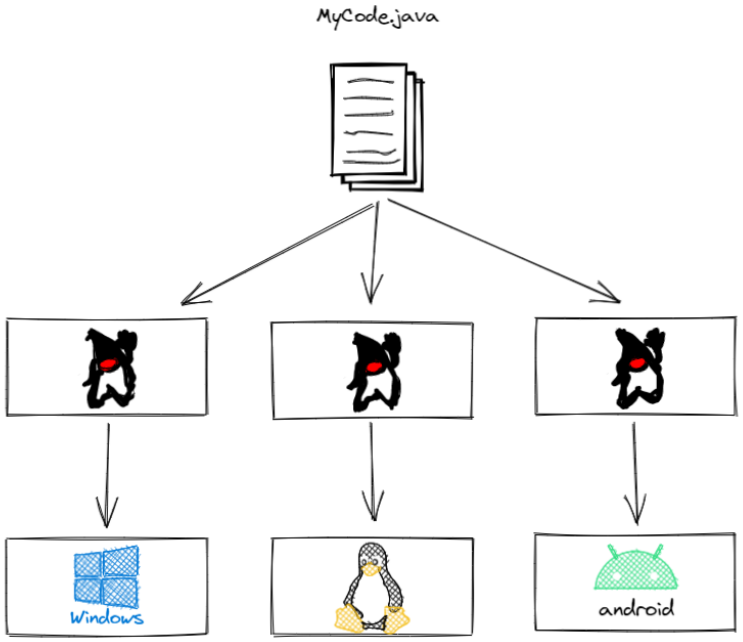
- First version was released in 1995
- Main design goals:
    - Platform for embedded devices
    - Portable (able to run on various targets)

java also had applets for web

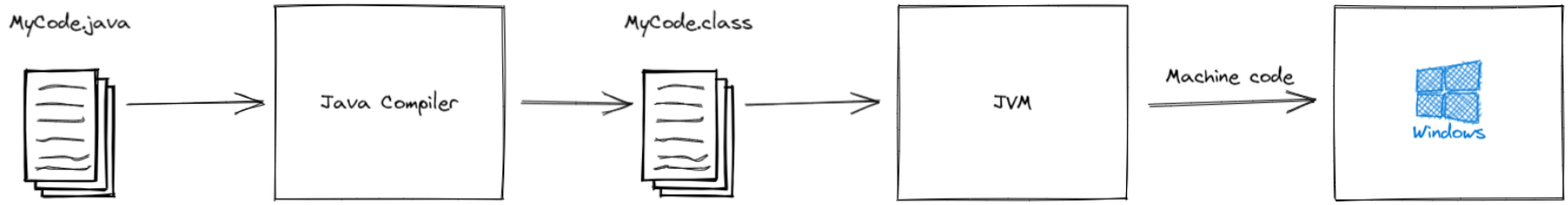
Write Code For Each Platform



Java Features: Write Once, Run Everywhere



Java Features: The JVM



Java Features: High Level Programming

Low Level Machine Code

```
mov     eax, 1200
mov     ebx, 5
add     aex, ebx
```

Unstructured Imperative Code

```
a = 1200
b = 5
a = a + b
```

Structured Imperative Code

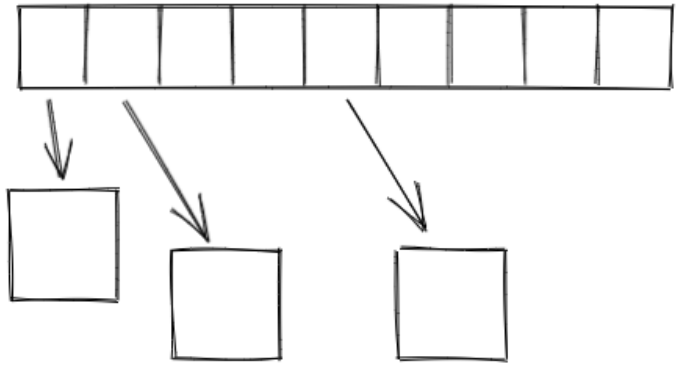
```
class Number {
  int add(int a, int b) {
    return a + b;
  }
}
```

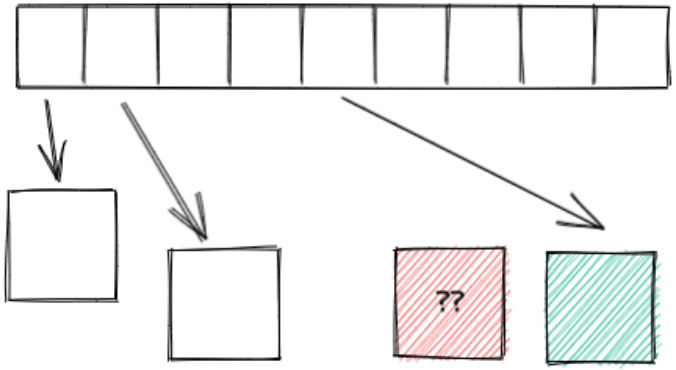
Java Features: Strong Typing

```
int a = 40;
String b = "Test";

a = b; // Type Error!
```

Java Features: Garbage Collection





Java Features: Object Orientation

- C: Just functions & structs
- C++: "Object Orientation" as grouping mechanism

- Java: Objects are "First Class Citizens"
    - Everything is an Object
    - Memory Allocation

Java Features: Security

- JVM built-in security features
    - Prevent arbitrary code execution
    - Proper use of privileged actions
- Supports TLS/SSL
- Supports code signing

Why Java? Who uses it?


Uber



NETFLIX








IJ

in

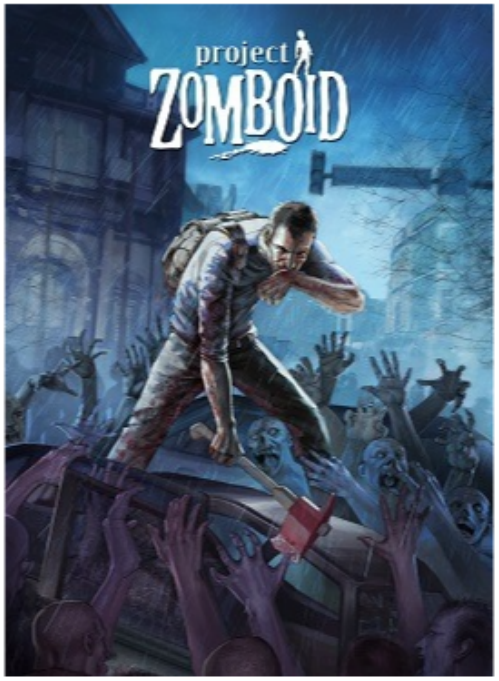
lyft

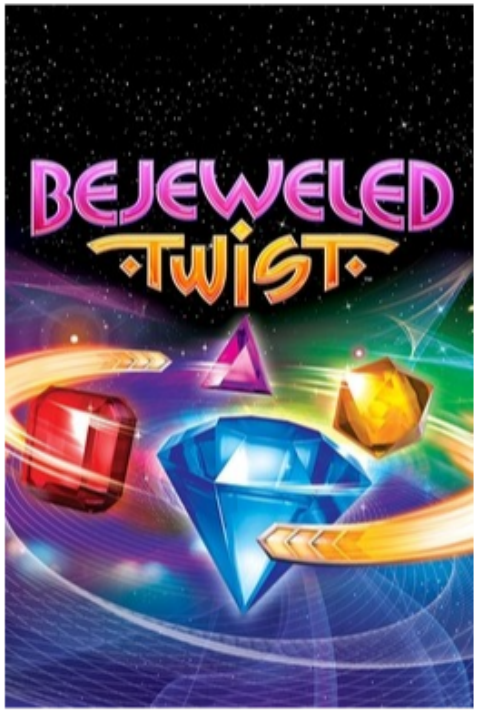

zalando

ebay

PIC
NIC

"Nobody makes games in Java!"



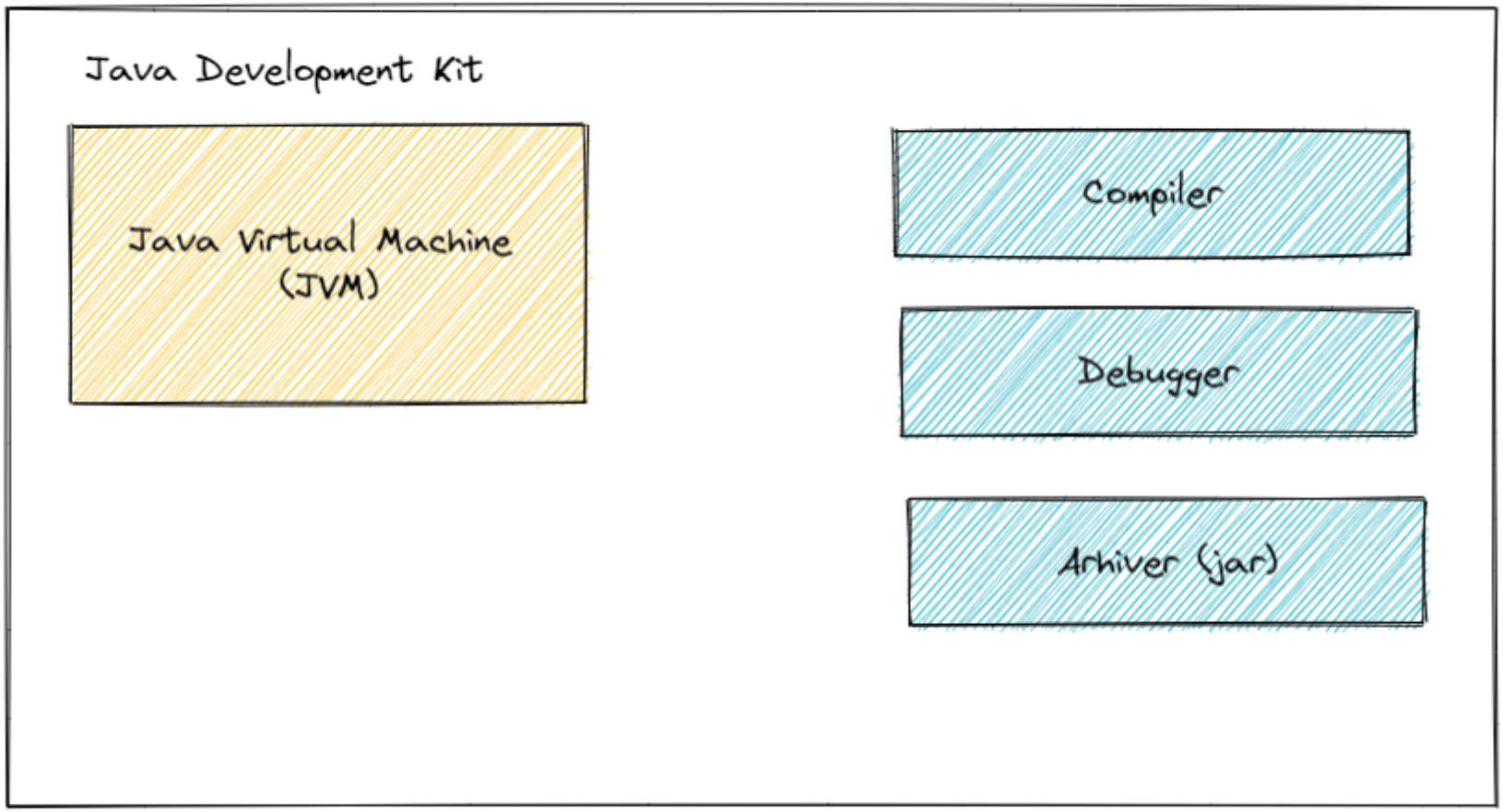


"Java is old and irrelevant!"

- Two new versions each year
    - Java is steadily evolving and catching up with the more modern and "fancy" languages
- JVM is as relevant as ever hosting plenty of other languages

java LTS releases are every 6 versions (each 3 years)

The Java Development Kit (JDK)



A Simple Java Program

```
public class HelloWorld
{
  public static void main(String[] args) {
```

```
    System.out.println("Hello World!");
  }
}
```

- `public class HelloWorld`

  Defines the class `HelloWorld`

- `public static void main(String[] args)`

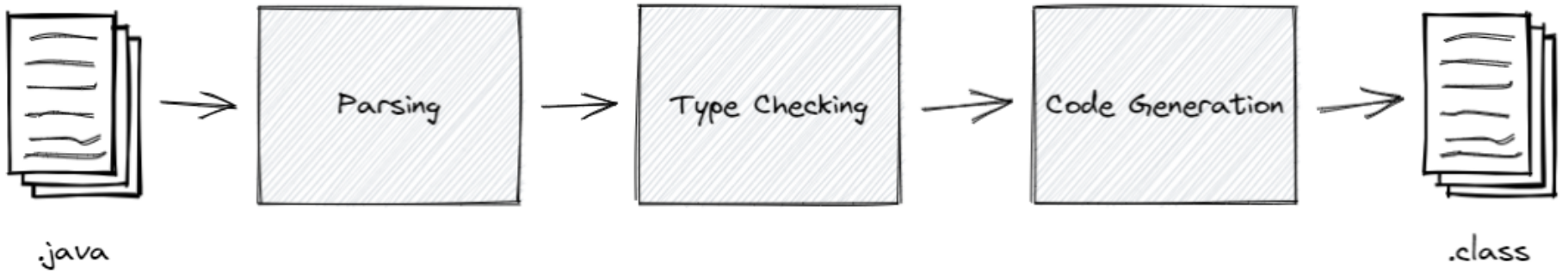
  Defines the `main` method

- `System.out.println("Hello World!");`

  Prints the message `Hello World!

Compiling the Source

- Java is a statically typed, compiled language



Detecting Compile-time Errors

- Mistakes in names
- Misspelled or misplaced keywords
- Assigning incorrect types

Detecting Compile-time Errors

- Spot the three errors:

```
public clazz BrokenWorld {

  int myNumber = "Hello!";

  public static void main(final String[] args) {
    System.out.println(myNummer + " world!");
  }
```

```
}
```

- What would happen if this code was run without checking?
- Can *all* errors be detected at compile-time?

Runtime Errors

- Spot the error:

```
void bigNumbers() {
  byte num = 120;
  System.out.println("num is above zero: " + (num > 0));
  num += 10;
  System.out.println("num is above zero: " + (num > 0));
}
```

- Why can this problem not be detected during compile-time?

Runtime Errors

- Even when the application is syntactically correct and type-safe, the compiler cannot account for all possible input / situations

> Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.
>
> — Rick Cook

Using `java` and `javac`

- Calling directly from the command line:

```
$ javac HelloWorld.java
$ java HelloWorld
Hello World!
```

- This is okay for small programs
- Impractical for complex applications with many dependencies

Managing Larger Applications

- Running Java’s tools on single files is doable
- Running them on 100+ files and dependencies is a nightmare

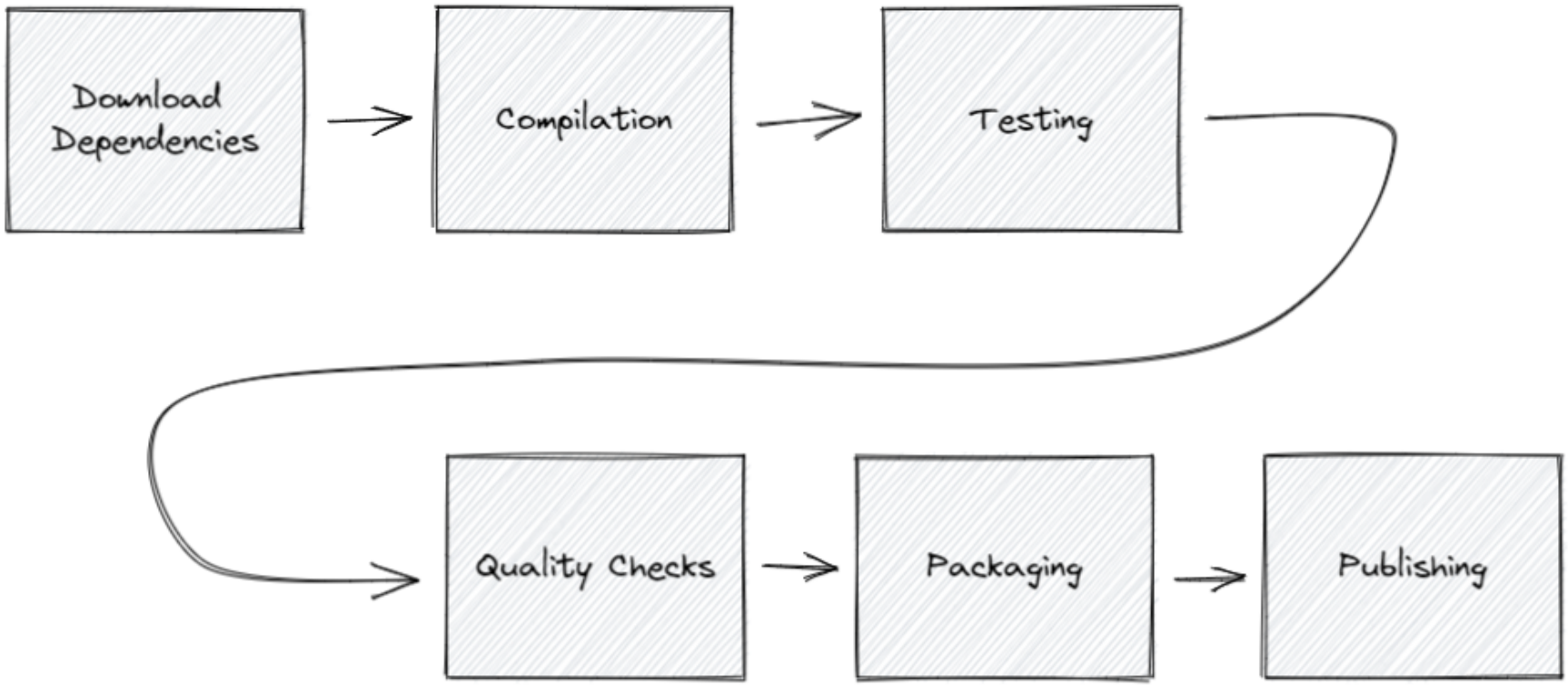
Tooling To The Rescue:

- Build Tools
- Dependency Management Tools
- Integrated Development Environments

Build Tools

- Supplies all source files and dependencies to `javac`
- Allow customization of build-process with plugins
- Dependency Management

Build Tools: Typical process



Integrated Development Environment (IDE)

- Combine Editing, Compilation, Running, Debugging
- Provides Syntax Highlighting and Code Templates
- Help with managing a large code base

- Integrate with Version Control
- Have rich plugin ecosystem

Which tools to pick!?

- Build tool is usually chosen by the client
- IDE comes down to personal choice
    - Look and Feel
    - Community
    - Pricing

Organizing Code

- Code is organized in Classes

- Classes are organized in packages
    - Packages group classes that logically belong together

```
+ src
+-+ main
  +-+ Main.java
    + database
    +-+ Database.java
    | | Queries.java
    | | DatabaseExceptions.java
    | + helpers
    | +-+ DatabaseTypeConvertors.java
    + services
    +-+ DataService.java
      | CalendarService.java
      | Utilities.java
```

Declaring and Using Packages

- A class *in* a package, must define a `package` at the top of its file

```
package database;

public class Database {}
```

- A class using classes from *another* package, must `import` that class

```
import database.Database;
```

```
public class Main {
  private Database myDatabase;
}
```

Naming Conventions

- Packages have names in lowercase
    - Reverse domain notation is used

      `nl.codenomads.courses.java.demo`

- Class names have names in CamelCase

  `ThisIsAVeryLongNameForAClass`

Naming Conventions Quiz

- What do you think these packages provide:
- `java.lang`
- `java.util`
- `java.util.concurrent`
- `java.crypto`
- `org.springframework.web.client`
- `org.springframework.scheduling`

java.lang - standard library, basic types and other fundamentals java.util - collections framework, date and time stuff, misc (base64) java.util.concurrent

- concurrent data structures and threading java.crypto - encryption and stuff

Primitive types

- Primitives are data types defined by Java
- Primitives hold data in a certain way
- Primitives have certain size

A primitive is a type, predefined by java. It can hold data in a certain way and for a certain size. There are only 8 different primitive types in java

Different Primitive types

byte	8 bit
short	16bit
int	32bit
long	64bit

Different Primitive types

float	32bit
double	64bit
boolean	1bit [1]
char	16bit

Boolean is VM dependent

Integer types

```
byte myByte = 127; // -128 to 127
short myShort = 32_767; // -32768 to 32767
int myInt = 2_000_000_000; // around 2 billion
long myLong = 9_000_000_000_000_000_000L; // A lot. Notice the 'L'!
```

Floats and Doubles

```
float largeFloatingPointNumber = 1.1e20f;
double largerFloatingPointNumber = 1.1e70;
```

Precision of Floats and Doubles

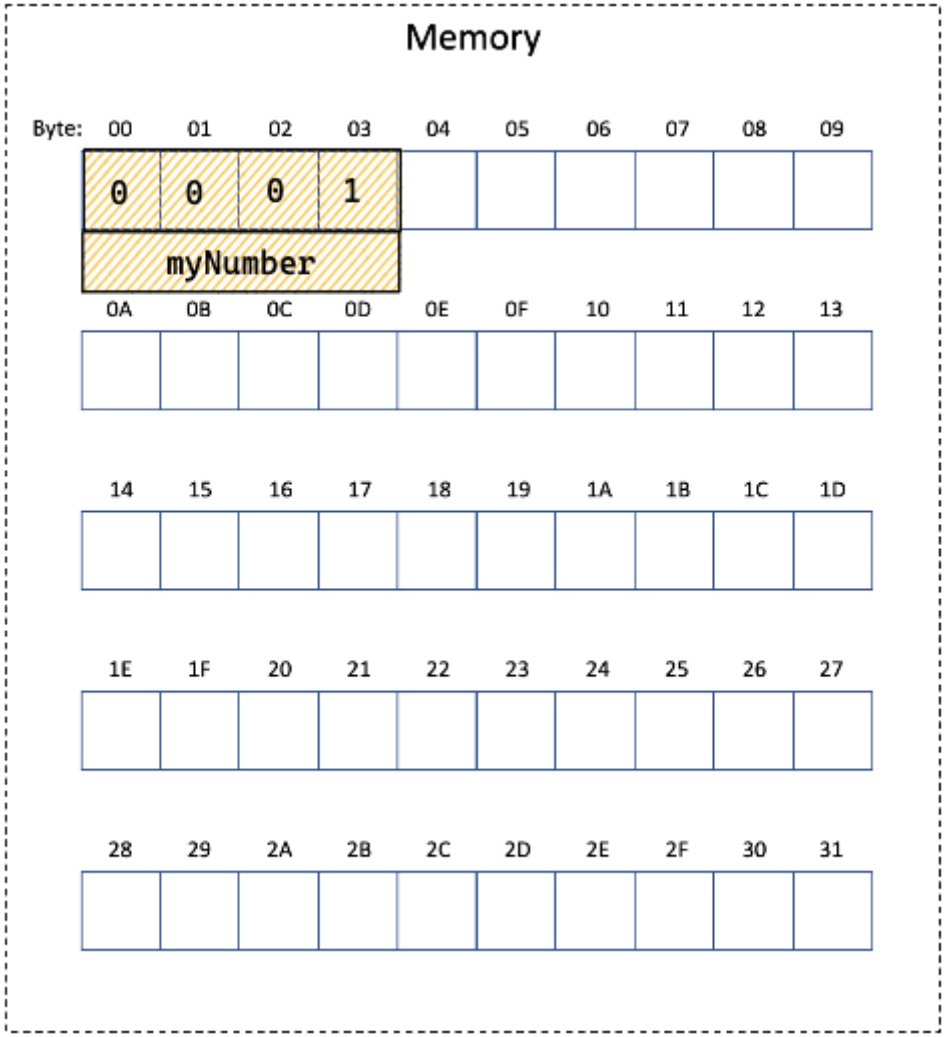
```
float largeFloatingPointNumber = 1.1e20f; // prints "1.1E20"
float largeFloatingPointNumber = 1.123456789e20f;  // prints "1.1234568E20" hmm..
```

Booleans and Chars

```
boolean myBoolean = true;
boolean myOtherBoolean = false;

char myChar = 'T';
```

Primitives stored in memory



This is how it could look in memory. We have a bunch of boxes, which are all bytes. Keep in mind that one byte can hold 256 different states, or in

numbers up to 255 including 0.

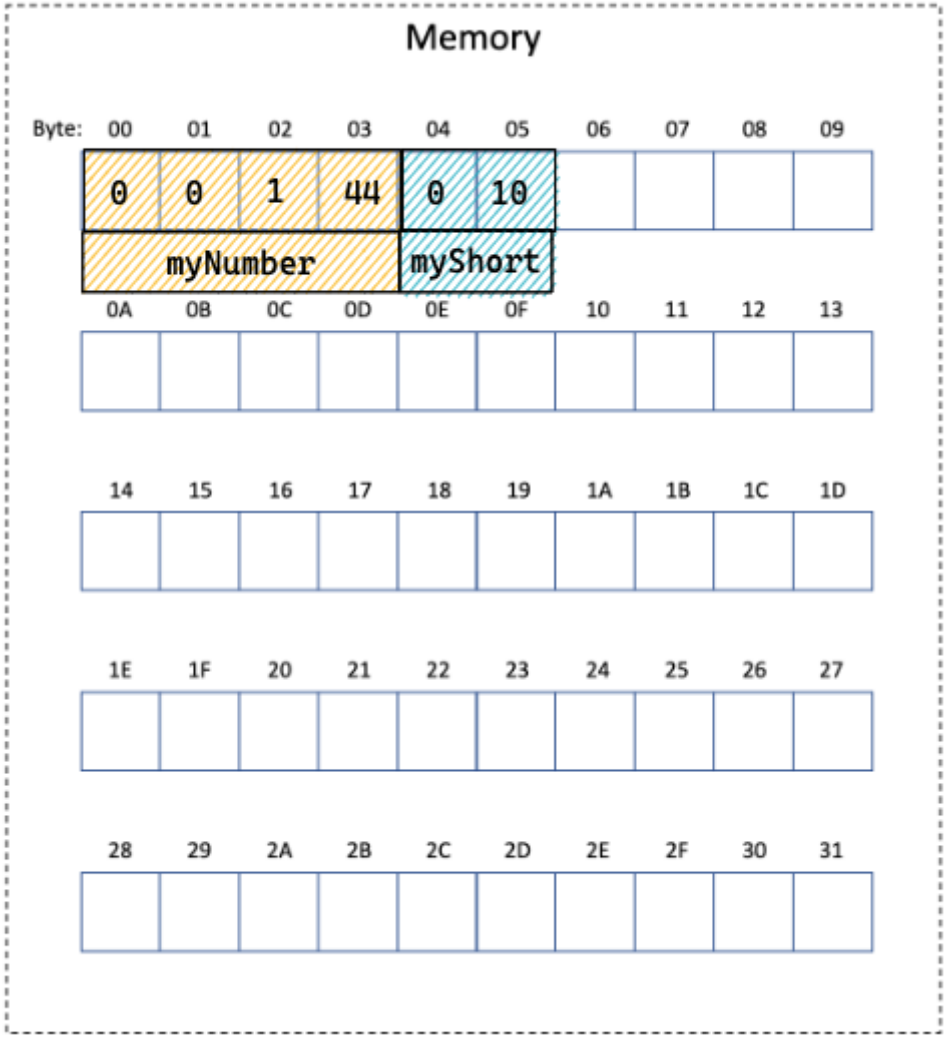
Also remember that an integers size is 32bits, or 4 bytes. That’s what you see here, four bytes are occupied by the myNumber variable.

Other primitives in memory

```
int myNumber = 300;
short myShort = 20;
```

Now lets change the code, how will this look in memory? Remember that a short takes 16 bytes, or two bytes

Multiple primitives in memory



We say that variable myNumber points to this memory address 00, and holds the value 300. We also say that the variable myShort points to this address 04 and holds the value 10.

Creating variables

```
[type] [variable_name];
```

```
 int myInt;
 float myFloat;
 boolean myBoolean;
 char myChar;
 // etc...
```

It is not necessary to assign a value to a variable. But what value do they hold if we don’t assign any ourselves? A default one! Depending on the type

Default values

```
// In some cases these values default to..
int myInt; // 0
float myFloat; //0.0
boolean myBoolean; //false
char myChar; // \u0000 (Not a readable character, refer to ASCII)
```

In some cases the Java compiler will automatically assign default values to variables if no value is assigned. Note that these values are actually the same as all 0s in memory! Generally it is best to always assign a value to a variable you create.

Setting values

```
 [type] [variable_name] = [value];
```

Setting values

```
// Integer-like primitives
byte myByte = 10;
short myShort = 100;
int myInt = 1000;
long myDouble = 10_000L;

// Floating point number primitives
float myFloat = 11.50f;
double myDouble1 = 1500.50d;
double myDouble2 = 1500.50;

boolean myBoolean1 = true;
boolean myBoolean2 = false;

char myChar = 'A'; // All characters possible
```

These are all different ways of setting values for primitives. Take note that some values need a suffix, like longs, floats, and optionally for doubles. Notice that the same number can be either a byte, short, or int in Java, depending on the primitive type it is assigned to.

Setting values from other variables

```
int myInt1 = 7;
int myInt2 = myInt1;

// What will be the output?
System.out.println(myInt2);
System.out.println(myInt1);
```

While using number and such to assign variables, we can also use already predefined variables to assign new variables. That is done exactly like this. What do you think the output of both variables will be?

Using variables

```
int result = 40 + 2;
```

The plus sign is something we have seen before, and I think you already know what it does. This is one of the operators that Java provides. There are many more, and we will go deeper into those later. For now lets stick to the plus operator. Here is the same operation but without using variables. It may seem obvious that the resulting type of this operation is an integer.

Adding different primitive types

```
short myShort = 40;
int myInt = 2;
[???] result = myShort + myInt;
```

But what about this? Java does support this, and favours certain types as a result of the + operator. In this case an integer type is the result.

Hands-on: different types and + operator

Let’s use the + operator on combinations of the following primitive types and find out what the result type is

```
 1: [???] result = 1L + 5L;
 2: [???] result = 1L + 5;
 3: [???] result = 1 + 0.0f;
 4: [???] result = 1L + 0.0d;
```

```
 5: [???] result = 1 + 'A';
 6: [???] result = 1L + true;
```

What are all the possible primitive types for these lines of code?

Results

```
1: [???] result = 1L + 5L; // long, float, double (long fits in double/float)
2: [???] result = 1L + 5; // long, float, double
3: [???] result = 1 + 0.0f; // float, double
4: [???] result = 1L + 0.0d; // double
5: [???] result = 1 + 'A'; // Trick question, 'A' is '65' in ASCII. Any primitive except boolean
6: [???] result = 'A' + 'B'; // Trick question, Any primitive except boolean and byte as the result is higher
than 127
7: [???] result = 1L + true; // Trick question, + and boolean doesn't work.
```

Storing text

```
// For the letter "A"
char letterA = 'A'
// For the word "hi"..?
char hello = 'H' + 'I'; // No no no!
```

char is for storing characters. But what about text? We create a text value with double quotes instead of single quotes. But what is the type called?

String

```
// For the word "hello"
```

```
String hello = "hello";
```

This type is called a String. It's called that way because it is a string of multiple characters. In here we can store text as long as we want. Note that we use a capital first letter for this type. This is because this is NOT a primitive type, but what we call an Object type. We will go much deeper into object later on

String basic operations

```
String hello = "hello";
String world = "world";

System.out.println(hello + " " + world);
```

Exercise

Write a program that takes two numbers and finds a number in between and prints the result as following:

```
The number in between is: [result]
```

[1] Not precisely defined (VM/implementation dependent)