

Intro to Java

CS2040S AY 19/20 Sem 1



Hello, World?

Welcome to CS2040S!

In this course, we will use **Java** to learn algorithms and data structures.

Don't worry though, if you aren't yet familiar with Java. This set of slides will aid you in transiting to Java.

That said, we cannot possibly cover everything about Java. You are expected to also learn more about the language on your own, or take CS2030 in parallel.

Assumptions

This set of slides is written with the assumption of CS1010/S or CS1101S knowledge.

That is, you should be (somewhat) familiar with C/Python/JavaScript and basic programming paradigms like variables, loops, if/else conditions.

You are not expected to be familiar with terminal commands. Note that terminal commands can be different for each OS. If you are unsure of any command, do a quick search on google.

Kattis

If you have not done so, sign up for a Kattis account at

<https://nus.kattis.com/register>

Remember to use your **NUS email**, and your name as per your **Matriculation Card**.

This will be the platform you will use for your problem sets. You can access them at https://nus.kattis.com/courses/CS2040S/CS2040S_AY_19-20_Sem_1

Let's get started!

Installation: Java JDK

Download and install Java Development Kit 8 (JDK 8) from <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

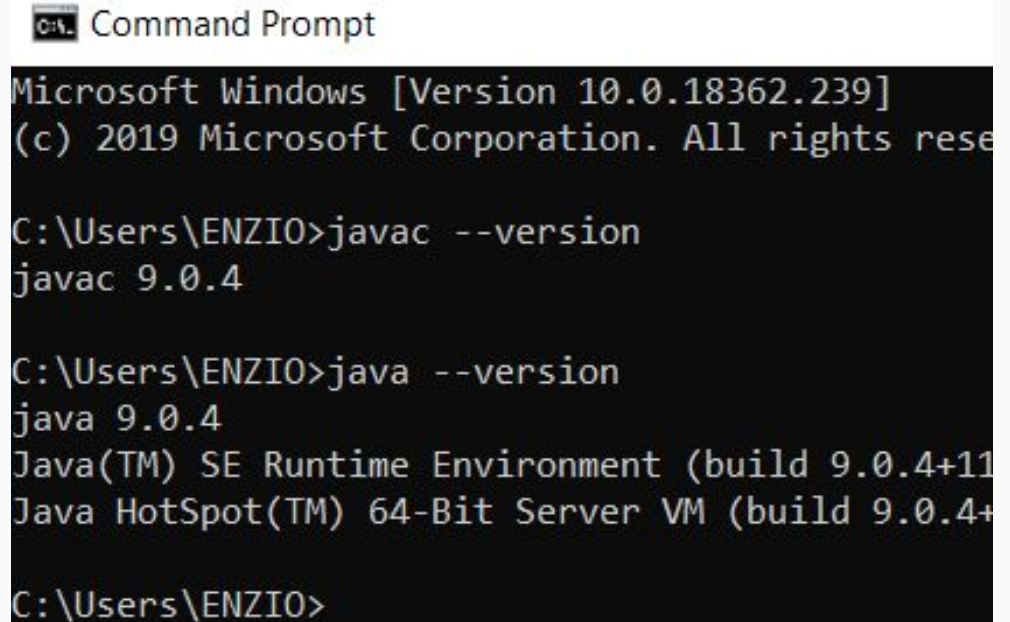
- a. Just installing Java (JRE) normally does not allow you to compile Java code. You'll need to install these Java binaries to be able to compile!
- b. You can also download other higher versions of the JDK.

Checking your Installation

Check to make sure that you have the JDK installed correctly by typing this into your terminal / command prompt:

javac --version

java --version



```
C:\> Command Prompt

Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ENZIO>javac --version
javac 9.0.4

C:\Users\ENZIO>java --version
java 9.0.4
Java(TM) SE Runtime Environment (build 9.0.4+11)
Java HotSpot(TM) 64-Bit Server VM (build 9.0.4+11)

C:\Users\ENZIO>
```

Writing your first Java program

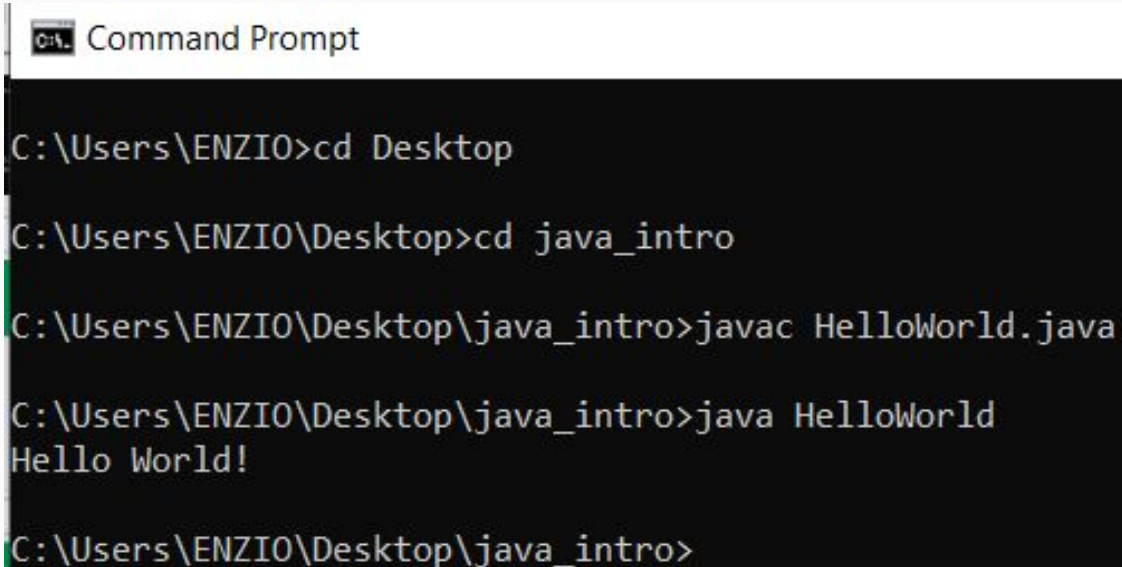
In any text editor such as Notepad++, Sublime Text, or Visual Studio Code, create a file called HelloWorld.java and type in the code below. (Take note of where you saved your file)

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


Hello World (examples on next slide)

- Navigate to the directory containing your file in the terminal.
 - Open up your terminal and type **`cd <folder containing file>`** to access the directory containing your file from the terminal.
 - Alternatively, you can use **`cd <folder>`** multiple times to access nested directories.
- Compile the HelloWorld.java file by typing **`javac HelloWorld.java`** in the terminal.
 - You will notice that there is a new HelloWorld.class file in the same directory.
- Type **`java HelloWorld`** to run your code.
 - "Hello World!" should appear in the terminal as output.

Hello World



```
Command Prompt

C:\Users\ENZIO>cd Desktop

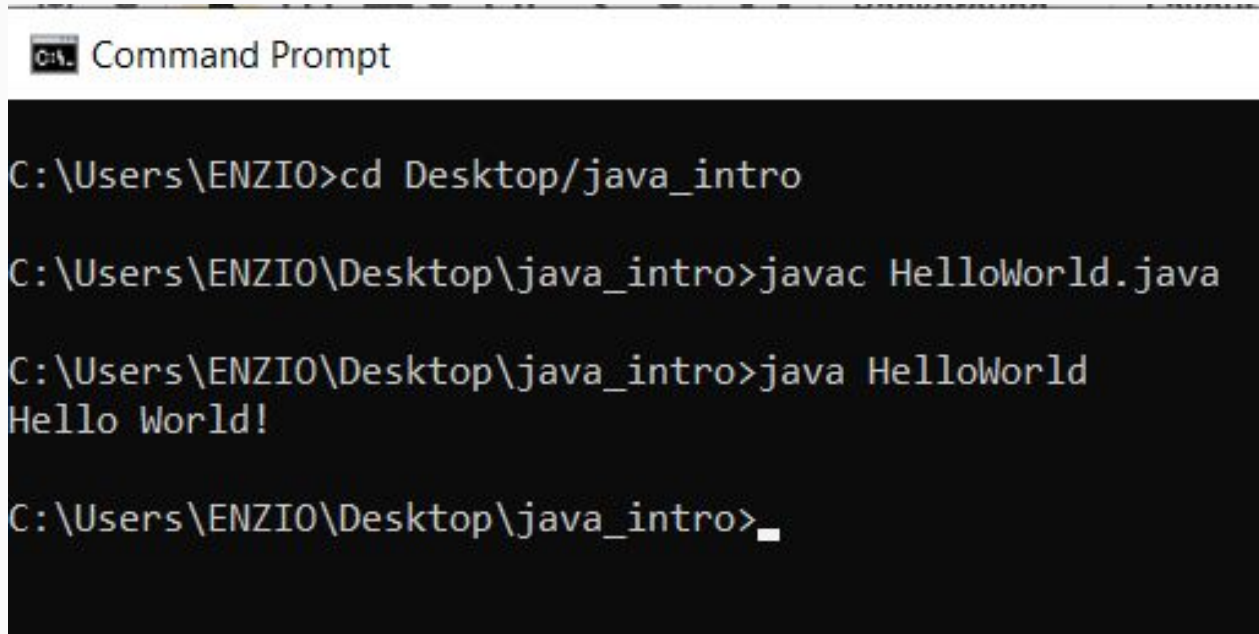
C:\Users\ENZIO\Desktop>cd java_intro

C:\Users\ENZIO\Desktop\java_intro>javac HelloWorld.java

C:\Users\ENZIO\Desktop\java_intro>java HelloWorld
Hello World!

C:\Users\ENZIO\Desktop\java_intro>
```

Hello World



```
C:\> Command Prompt

C:\Users\ENZIO>cd Desktop/java_intro

C:\Users\ENZIO\Desktop\java_intro>javac HelloWorld.java

C:\Users\ENZIO\Desktop\java_intro>java HelloWorld
Hello World!

C:\Users\ENZIO\Desktop\java_intro>_
```

Hello Kattis

- Try submitting the HelloWorld.java file to <https://nus.kattis.com/sessions/c92h2p/problems/hello>
- You should see something like this

ID	DATE	PROBLEM	STATUS	CPU	LANG
TEST CASES					
3336526	2018-10-28 21:32:38	Hello World!	✓ Accepted	0.05 s	Java
✓					

- Other possible results include: *Time / Memory Limit Exceeded, Compile Error, Runtime Error, Wrong Answer*

Jshell (> Java SE 9)

Jshell have the capabilities of running java interactively (REPL)

Run and tinker with java without the hassle of compiling and re-running the program.

Type *jshell* into your terminal to try it out.

```
doe:~ john$ jshell
| Welcome to JShell -- Version 10.0.2
| For an introduction type: /help intro
```

```
jshell> class Point {
...>     public int x;
...>     public int y;
...> }
| created class Point
```

```
jshell> Point p = new Point()
p ==> Point@239963d8
```

```
jshell> p.x = 5
$3 ==> 5
```

Java Syntax

Java Syntax

Java is strict on its syntax.

As opposed to Python or Javascript, Java will refuse to compile if you're missing a semicolon.

Unlike Python, Java doesn't care about indentation and whitespace.

However, your Prof and TAs care (and you should). We (may) need to read your code!

Declaring variables

Java, like C, is also a strictly typed language.

Python	Javascript	C	Java
<code>i = 3</code> (semicolon optional)	<code>var i = 3</code> (semicolon optional)	<code>int i = 3;</code> (note: semicolon)	<code>int i = 3;</code> (note: semicolon)
<code>s = "Hello"</code>	<code>var s = "Hello"</code>	<code>char *s = "Hello";</code> (C has no built-in String)	<code>String s = "Hello";</code>
<code>i = "World"</code> (previously declared)	<code>i = "World"</code> (previously declared)	Syntax Error. (Cannot int → String.)	Syntax Error. (Cannot int → String.)

Data Types

Here are some of the commonly used data types:

Name	Description	Name	Description
<code>int</code>	Integers. Whole numbers. (From -2^{31} to $2^{31} - 1$)	<code>long</code>	Integers with much larger range (From -2^{63} to $2^{63} - 1$)
<code>float</code>	Floating point numbers.	<code>double</code>	More precise floating point numbers
<code>char</code>	Characters: a, b, c	<code>String</code>	A bunch of characters.
<code>boolean</code>	<code>true</code> or <code>false</code> . Note: <code>true/false</code> is not capitalised in Java.		
<code>void</code>	Not valid for initialising variables. Only used as return types to functions where you want to declare that “this function does not return anything”.		

Declaring variables

Some rules (there are more):

1. Must declare a variable type for every variable upon initialisation.
 - a. “**var**” -- only supported from Java 10 onwards.
2. Cannot change this variable type in the future anymore.
3. Cannot assign value of wrong type.

Casting variables

Casting is converting from one data type to another. E.g. Convert an Integer to a Long. Convert a Double to an Integer.

1. Generally, casting can only be done from a more restrictive type to less restrictive type. The other way around requires force type cast. e.g.
 - a. `int a = 5; long b = a;` (OK).
 - b. `long a = 5; int b = a;` (Compile Error). → `long a = 5; int b = (int) a;` (OK)
2. Always OK to cast from a sub-class to its super-class.
 - a. We will cover this shortly when we introduce “classes”.

Math

Max range for int: -2^{31} to $2^{31} - 1$ / long: -2^{63} to $2^{63} - 1$.

What happens if we exceed this range? OVERFLOW!

2147483646, 2147483647, -2147483648, -2147483647, ...

What if we need numbers greater than 2^{63} ? Use Java BigInteger class.

```
BigInteger num = new BigInteger("100000000000000000000000000000000");
```

Note: Python has no such concept of Overflow because numbers will be automatically “BigIntegers”.

Math

Addition: $10 + 20 = 30$

Subtraction: $20 - 10 = 10$

Multiplication: $20 * 20 = 400$

Division: $59 / 20 = 2$. (Always integer division / **truncates**)

Floating division: (double) $59 /$ (double) $20 \rightarrow 2.95$ (Cast to double first)

Power: `Math.pow(double a, double b)` \rightarrow returns a^b as a **double**. No `**` operator.

Modulo: $59 \% 20 = 19$. $-59 \% 20 = -19$ \leftarrow returns the negative version

Math operations follow BODMAS rules.

Loops

Basic for-loop structure:

```
for (int i = 0; i < 1000; i++) {  
    ...  
}
```

Similar to C/JS (note `int i`, not `var i`).

Python: there's no `for i in range(x)` in Java :(

There's also the enhanced for-loop `for (Integer x : array) {...}`, but we'll cover this when we touch Arrays/ArrayLists later on.

Loops

Basic for-loop structure:

```
for (int i = 0; i < 1000; i++) {  
    ...  
}
```

red: Initialisation. Runs once, for you to initialise anything you need.

white: Condition. The for loop will exit (“break”) when the condition is false.

blue: Afterthought. What happens at the end of each iteration.

All three sections are optional. E.g. you can have `for(;;) { ... }` which is an inf loop.

Loops

Basic while-loop structure:

```
int i = 0;
while (i < 1000) {
    ...
    i++;
}
```

Similar to C/JS/Python, but Python users: note the use of braces.

Comments

It's always good to let your readers (and yourself) know what you're writing.

You can add comments in Java using double slash for a line, or `/* ... */` for a block spanning multiple lines.

```
// This for-loop will loop exactly 999 times.  
for (int i = 1; i < 1000; i++) {...}
```

Conditions

Same as C. Python note: There's no `elif` (Python). Use `else if`.

My wife said: "Please go to the store and buy a carton of milk and if they have eggs, get six." I came back with 6 cartons of milk. "Why did you get 6 cartons of milk?" "Because there were eggs."

```
int milk = 1;
if (hasEggs) { milk = 6; }
else if (noMoney) { milk = 0; }
else { ... }
```

Methods / Functions

Syntax:

```
ReturnType methodName(VarType1 varName1, ... ) {  
    // Do Something  
}
```

Example:

```
int square(int num) {  
    return num * num;  
}
```

Java OOP (brief introduction)

Java OOP (brief introduction)

Java uses the Object-Oriented Programming paradigm. You can best think of this as a higher-level “function”.

Where a function encapsulates one instruction (e.g. move a shape by distance x), a class encapsulates the entire object (e.g. the shape itself) and comprises many functions.

Java OOP (brief introduction)

Almost everything in Java is an Object! (It literally subclasses Object).

Remember we did the Hello World example at the start?

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

HelloWorld is a class. And the special main(String[] args) function is a method inside the HelloWorld class. You can only **java <ClassName>** .class files with a main method.

Your own classes

You can create your own classes!

```
class Shape{  
    public int size = 5; // 5px  
}
```

```
class Square extends Shape { // Square is a shape, and more!  
    public int area = size * size;  
    // Can use variables from super-class  
    // if it's public  
}
```

Java I/O

In CS2040S, we will use the Kattio class provided by Kattis to help us read input and write output.

It can be downloaded here:

<https://open.kattis.com/download/Kattio.java>

(There are other I/O methods such as Scanner, System.out, PrintWriter, BufferedReader etc. We do not have time to cover everything, you should explore them on your own.)

Java I/O

```
public static void main(String[] args) {  
    Kattio io = new Kattio(System.in, System.out);  
}
```

This creates a Kattio object that can read stuff from the standard input, where you enter your commands/input for a task.

For example, you can read an Integer by doing

```
int n = io.getInt();
```

Java I/O

To print to the standard output, use

```
io.print(...); //Prints stuff  
io.println(...); //Prints stuff, then adds a new line.
```

Or if you're more familiar with C,

```
io.printf("format", variables);
```

The print/println is very convenient, while printf is more flexible as it allows you to format your output (e.g. try to print this: "Input Set 1: My answer is 5").

Java I/O

```
io.printf("format", variables);
```

```
int numCakes = 12;
```

```
io.printf("I have %d cakes.\n", numCakes);
```

```
//I have 12 cakes.↵
```

```
double money = 3.5799999;
```

```
io.printf("I have %.2f dollars.\n", money);
```

```
//I have 3.58 dollars.
```

Format	Type
"%s"	String
"%d"	Int / long
"%f"	float / double

Java I/O

Important! At the end of your program, always run the flush method once.

```
io.flush(); //Flushes the output buffer
```

Analogy: This 'delivers' the output that your program has produced to Kattis for grading.

Java's built-in classes

There are many more classes, built in to Java, that you'll soon encounter.

For example, Java has an `ArrayList` class that works similarly to `Arrays`¹ in C, or `List` in JS/Python.

You can initialise an `ArrayList` with this line:

```
ArrayList<Integer> arr = new ArrayList<Integer>();
```

This creates a new, empty, arraylist that can hold (only) `Integers`.

¹Arrays in C are fixed-sized. Fixed size Arrays do exist in Java too. You can think of `ArrayLists` as dynamically-sized "arrays".

Variables, part 2

So far, we've been using Integer and int very interchangeably. In Java, int is a “primitive type”, while Integer is a class that encapsulates this int.

We previously said that almost everything is an object. It turns out that these primitive types `int`, `double`, `char` etc, are not objects.

`ArrayList` for example, can only hold objects, so to work around this issue, the Integer class is essentially a wrapper around an int. So while `ArrayList<int>...` is illegal, you can simply do `ArrayList<Integer>...` and Java will handle the work for you.

Casting, part 2

We briefly mentioned previously that you can “cast” from a more restrictive to a less restrictive class.

```
class Shape { ... }  
class Square extends Shape { ... }
```

```
Square sq = new Square(5); //new square with side length 5.  
Circle c = new Circle(5); // new circle with radius 5.  
Shape x = sq; // Can “cast” a square into a shape.  
Shape y = c; // Can also “cast” a circle into a shape.
```

Casting, part 2

This is useful if we have a bunch of different shapes. We cannot force fit a circle into a square, but we can create an arraylist of shapes, since all of them are shapes.

```
ArrayList<Shape> list = new ArrayList<Shape>();
```

```
list.add(x);
```

```
list.add(y);
```


Basic Style Guide

Basic Style Guide

Generally, it's always good to conform to certain styles so that it makes it easier for others to read and understand your code. While everyone has their preferences, here are some basic guides that you should try to follow as much as possible when coding.

Basic Style Guide

1. Be consistent.

- a. For example, it's fine if you use 2 spaces, 4 spaces, or tab when indenting. Just be consistent. Make sure that your indentation stays consistent (don't use 2 spaces then change to a tab or something), and make sure you do indent your code for readability.
- b. Another common one is the braces. Some may wish to put their opening brace on a new line, which is fine, but do be consistent.

2. Name your variables well

- a. Instead of `int a, b, c, d, e` which makes it hard to understand what the variable is for, try to name it something more meaningful. For example, `int bucketsOfWater`.
 - i. It's fine to use one-letter variables if it's a counter for example in a loop.
- b. In Java, the convention is to use `lowerCamelCase` for variable and method names (e.g. `int numberOfStudents = -1;`), and `UpperCamelCase` for class names (e.g. `class HelloWorld`).

Basic Style Guide

3. Write comments to help yourself.
 - a. After a while, you may forget what you've been doing, so write comments to help out your future self! (and anyone else reading your code)
4. Try to use OOP and functions instead of repeating yourself
 - a. In general, try to follow the DRY (Don't Repeat Yourself) principle and encapsulate part of the logic where possible. You may find this a little tedious at the start, but it'll come in very handy when you implement more advanced data structures.

Recap

- Introduction to Java Syntax
- Java OOP and I/O
- Introduction to Kattis

Problem Set 0 (optional) is released on Kattis. Practice some problems!

That's all folks!

All the best for CS2040S!