Course Overview and Java Basics

Table of contents

Today's outline 1
Programming in the large 2
Java overview 2
Data types in Java 3
Scope 4
Control flow: sequence, selection, and iteration 4
Java arithmetic 6
Summary 6
Reminders 6

See csse2002[1] for staff, assessment, and course logistics — this note covers the technical content from Lecture 1 (the live lecture; there's also a **recorded** lecture this week, see 2026-02-26-java-basics-part-02-collections-and-strings[2]).

Today's outline

- Programming in the large vs. programming in the small — why this course isn't just "a Java course"
- Java vs Python: compiled vs interpreted, statically vs dynamically typed
- Java's primitive and reference types
- Scope
- Sequence, selection, and iteration in Java
- Java arithmetic

[1] courses/csse2002/csse2002.pdf
[2] courses/csse2002/2026-02-26-java-basics-part-02-collections-and-strings.pdf

Programming in the large

Code from an intro course (CSSE1001/ENGG1001) has likely been small, written solely by you, for a limited purpose, and to exist for a limited time — but large software projects are usually none of these. CSSE2002 teaches the individual discipline and programming practices needed to write software suitable for integration with large software systems: documenting, debugging, and testing code so that programming scales.

This is not just a Java course — Java is the vehicle, not the destination.

Java overview

Java is a general-purpose programming language that is:

- **Compiled**
- **Statically typed**
- **Object-oriented**
- **Memory safe**

Java vs Python: compiled vs interpreted

Python is *interpreted*: you run the Python interpreter directly on the source.

```
$ python hello_world.py
```

Java is *compiled*: the source is first compiled to bytecode, then run on the JVM.

```
$ javac HelloWorld.java
$ java HelloWorld
```

Homework (from the slides). What is the benefit of having bytecode? What are the differences between bytecode and machine code?

Java vs Python: static vs dynamic typing

Python is *dynamically* typed — whether an appropriate type is used is determined when the program runs. Java is *statically* typed — types are checked for every scenario at compile time. Static-typed languages require an explicit type declaration for every piece of data (variable, parameter, return value); dynamic languages instead infer (or guess) the type in use.

```
>>> x = 1
>>> print(x, type(x))
1 <class 'int'>
>>> x = 1.7
>>> print(x, type(x))
1.7 <class 'float'>
>>> x = "Hello"
>>> print(x, type(x))
Hello <class 'str'>
```

Python doesn't restrict a variable to one type over its lifetime, even though it tracks each value's own (runtime) type. Java is the opposite:

```
int x;
x = 1;      // OK
x = -5000;  // OK
x = 1.0;    // illegal: not a whole number
x = "15";   // illegal: cannot convert from a string to a number
```

Python also uses indentation to delimit code blocks, while the Java *compiler* uses `{}` to delimit blocks and `;` to terminate statements.

Data types in Java

See java-primitive-and-reference-types[3] for the full primitive-vs-reference-types table introduced in this lecture. In short: Java distinguishes **primitive types** (built-in, fixed representations like `int` and `boolean`) from **reference types** — everything else, i.e. classes (including `String` and arrays).

[3] courses/csse2002/java-primitive-and-reference-types.pdf

Scope

A variable's scope is where it can be used — in Java, a variable's scope ends at the end of the block in which it's declared.

Question. What happens when this executes?

```java
public static void main(String[] args) {
    if (5 > 4) {
        int z = 2;
    } else {
        int z = 3;
    }
    System.out.println(z);
}
```

This is a compile error: `z` is declared inside the `if/else` blocks, so it's out of scope by the time `System.out.println(z)` runs.

Answer. Declare the variable inside the same (or a higher) block as where it's used — e.g. declare `z` before the `if/else` (outer scope), or move the `println` inside each branch.

Control flow: sequence, selection, and iteration

Sequence

- Each line must end with a semicolon (`;`).
- Declarations must come before other assignments.
- Instructions are executed sequentially.

Methods are Java's equivalent of Python functions — but all code must live inside a class:

```python
def add_two_numbers(num1, num2):
    result = num1 + num2
    return result
```

```java
class Arithmetic {           // All code must be inside a class.
    int add_two_numbers(int num1, int num2) {
        int result = num1 + num2;
        return result;
    }
}
```

Calling a method looks the same as calling a Python function: `add_two_numbers(10, 20)`.

Selection: if / switch statements

Java `if`/`else` and `switch` work like their Python counterparts, just with Java's block/statement syntax (`{}`, `;`, and `case`/`break` for `switch` rather than Python's `match`).

Exercise. Write a method `printGrade` that takes a student's mark and determines their grade:

Mark	Grade
83.00 – 100.00	High Distinction
73.00 – 82.99	Distinction
63.00 – 72.99	Credit
50.00 – 62.99	Pass
0.00 – 49.99	Fail

Iteration: while and for loops

A `while` loop has an **initialiser** (set up a counter), a **test/condition** (check the counter), and an **update** (increment/decrement the counter):

```java
int i = 0;
while (i < 10) {
    System.out.println(i);
    i++;
}
```

```python
i = 0
while i < 10:
    print(i)
    i += 1
```

A `for` loop bundles all three parts together:

```java
for (int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

```python
for i in range(10):
    print(i)
```

Exercise. A factorial of a number n, written $n!$, is the product of all numbers less than or equal to n. Write a method `factorial` that calculates it.

Exercise. For all positive integers, $n! = n \times (n-1)!$. Write an equivalent `factorialRec` method that uses recursion to calculate the result.

Java arithmetic

- `+`, `-`, `*` work the same way as Python — **except** integer division: `/` between two `int`s in Java truncates towards zero (integer division), rather than always returning a float like Python's `/`.
- Comparison and logical operators are semantically identical to Python, just different syntax: `==`, `!=`, `<`, `>`, `<=`, `>=` are the same spelling, but Java's logical operators are `&&`, `||`, `!` where Python uses `and`, `or`, `not`.
- Augmented assignment (`+=`, `-=`, `*=`, `/=`) and increment/decrement (`++`, `--`) work as in many C-like languages — increment/decrement only apply to primitive types.

Summary

- The entry point for all Java programs is the `main` method: `public static void main(String[] args)`.
- Java is a compiled and statically typed language, compared to Python.
- Most arithmetic operations in Java function identically to Python.
- The same core principles of sequence, selection, and iteration apply to Java, but the syntax is different.

Reminders

- Your first Ed Lessons exercise is due by 1pm next Wednesday.
- Applied and practical classes start next week (Week 2).

Tasks this week:

- ☐ Attend the lecture.
- ☐ Week 1 Ed Lessons exercises.
- ☐ Ensure you can access Ed discussion.
- ☐ Enrol in your contact and practical classes.
- ☐ Watch the recorded lecture on Collections — see 2026-02-26-java-basics-part-02-collections-and-strings[4].
- ☐ Install IntelliJ IDEA & Java Development Kit (JDK) 21.

[4] courses/csse2002/2026-02-26-java-basics-part-02-collections-and-strings.pdf