CSSE2002 — Programming in the Large

Table of contents

Overview 1
Assumed background / prerequisites 2
Staff 2
Course structure 2
Lectures 3
Learning outcomes 5
Assessment 5
Grading cutoffs 7
Academic integrity 7
How to learn (course philosophy) 7

Overview

Working on large and complex software systems, and keeping those systems maintainable, requires disciplined individual practices: software must be well-specified, well-implemented, and well-tested. This course covers concepts and techniques in modern programming languages that support good practice — OO concepts, genericity, and exception handling — with specific application to file IO and GUIs, taught in **Java**.

Higher-level language constructs (classes with robust, compact interfaces) are needed to manage the complexity of large software systems. The course uses Java to introduce object-oriented programming, data abstraction, design, refactoring, and unit testing, aimed at making an individual's code contributions fit for integration into a larger codebase.

What this course isn't: not just a Java-syntax course — knowing Java is a means to the actual goal of individual programming discipline at scale.

Assumed background / prerequisites

- Prerequisite: CSSE1001 or ENGG1001 (or equivalent). Incompatible with COMP2500, COMP7908, CSSE7023.
- Assumed Python skills: variables; control flow (for/while/if); defining and calling functions; recursion; fundamental data structures (arrays, lists, strings, dictionaries); instantiating and defining classes; inheritance.

Staff

- **Dr Thilina Halloluwa** — course coordinator & lecturer. HCSE (human-centred software engineering) researcher; ~15 years as an academic across UOC, QUT, USYD and UQ; product development specialist.
  - Personal: t.halloluwa@uq.edu.au · course: CSSE2002@eecs.uq.edu.au
  - Office **78-318**; consultation **via email**.
- A team of casual academics (a dozen-plus) run the applied and practical classes.

Course structure

- **2 units** (~10 hours/week).
- **Lecture** (2 hours/week) — theory and demonstrations. Lectures begin in Week 1.
- **Applied Class** (1 hour/week) — theory exercises; critical for the final exam. Starts Week 2.
- **Practical** (2 hours/week) — programming practice; bring your own laptop. You can get help, but assessment must be solo work. Starts Week 2.
- **CodeCamp sessions** (~30 min) — part of the contact hours, alongside the lecture, applied class and practical. There is also a CodeCamp video playlist of supplementary material and a Git repo of code samples.
- **Ed Lessons** — weekly Java programming exercises in Java, released a week ahead and due the following Wednesday 1pm (the "Problem Sets" assessment item below). Ed is also where course discussion happens.

Make sure you have signed up for your applied and practical classes; contact signon@eait.uq.edu.au if you have problems.

Teaching philosophy

The course is built around **independent learning** — muddy cards, HW cards, and CodeCamp sessions — backed by **student support**. A HW card is a homework question to take away, set

at the end of a lecture section. The maxim behind the approach: *"I hear and I forget. I see and I remember. I do and I understand."*

Lectures

Week	Lecture
1	2026-02-26-course-overview-and-java-basics[1] · 2026-02-26-java-basics-part-02-collections-and-strings[2]
2	2026-03-05-object-oriented-programming-i[3]
3	2026-03-12-object-oriented-programming-ii[4] (+ Specification)
4	2026-03-19-exceptions[5] (+ Specification) · supplementary: java-version-control[6]
5	2026-03-26-testing[7]
6	2026-04-02-refactoring[8] (+ cohesion & coupling, SOLID I)
7	2026-04-16-solid-principles-ii[9]
8	2026-04-23-java-io[10]
9	2026-04-30-generics-in-java[11]
10	*no lecture*
11	2026-05-14-lambdas-streams-and-events[12]
12	2026-05-21-correct-programming[13]
13	2026-05-28-course-summary-and-exam-preparation[14]

Weeks 3 and 4 both carry a Specification section, and week 6 is really three topics (refactoring, cohesion & coupling, and the first two SOLID principles). SOLID I is part of the week 6 refactoring lecture, which is why there is no separate "SOLID Principles I" lecture.

[1] courses/csse2002/2026-02-26-course-overview-and-java-basics.pdf
[2] courses/csse2002/2026-02-26-java-basics-part-02-collections-and-strings.pdf
[3] courses/csse2002/2026-03-05-object-oriented-programming-i.pdf
[4] courses/csse2002/2026-03-12-object-oriented-programming-ii.pdf
[5] courses/csse2002/2026-03-19-exceptions.pdf
[6] courses/csse2002/java-version-control.pdf
[7] courses/csse2002/2026-03-26-testing.pdf
[8] courses/csse2002/2026-04-02-refactoring.pdf
[9] courses/csse2002/2026-04-16-solid-principles-ii.pdf
[10] courses/csse2002/2026-04-23-java-io.pdf
[11] courses/csse2002/2026-04-30-generics-in-java.pdf
[12] courses/csse2002/2026-05-14-lambdas-streams-and-events.pdf
[13] courses/csse2002/2026-05-21-correct-programming.pdf
[14] courses/csse2002/2026-05-28-course-summary-and-exam-preparation.pdf

The full weekly plan

This is the plan for the whole semester as published in Week 1, including what each **applied class** and **practical** covers. It differs in places from what was actually delivered (e.g. Lambdas & Streams came in week 11 rather than on 07 May); the *Lectures* table above is the one that matches the notes.

Week	Lecture date	Lecture topic	Applied class	Practical
1	26th Feb	Course Overview and Java Basics	*recorded lecture on Strings and Collections*	*(as applied)*
2	05th March	Object Oriented Programming Day 01 + Modeling	Java Basics	Java Basics
3	12th March	Object Oriented Programming Day 02 – Specification	Stack & Heap	Java Classes
4	19th March	Exception Handling	Method Dispatch & Casting	**Practice Practical Demonstration**
5	26th March	Testing with JUnit	Exception Handling	Debugging
6	02nd April	SOLID Principles | Refactoring	Blackbox and Glassbox Testing	**Assignment Help**
—	*Mid Semester Break*			
7	16th April	SOLID Principles	Class Invariants	**Practical Demonstration**
8	23rd April	Java I/O	Substitution Principle	JUnit Testing
9	30th April	Generics	Coupling and Cohesion	Refactoring
10	07th May	Lambdas and Streams	Dependency Inversion	Java I/O
11	14th May	Guest Lecture	Lambdas and Streams	**Assignment Help**

Week	Lecture date	Lecture topic	Applied class	Practical
12	21st May	Event Driven Programming	*no class*	Generics
13	28th May	Course Summary & Exam Preparation	Practice Exam	Practice Exam

Four practicals are bolded above: the practice demonstration in week 4, assignment help in week 6, the real **Assignment 1 Practical** demonstration in week 7, and assignment help again in week 11. Only the week-7 one is the assessed A1 Practical. The plan labels it just "Practical Demonstration"; matching it to the A1 Practical window (13–17 April) is an inference from the assessment dates below.

What is not examinable

Confirmed in Week 13:

- **Version Control** (week 3-4 supplementary)
- **Event Driven Programming & GUIs** — the events half of week 11; lambdas *are* examinable
- **Correct Programming** (week 12)

Learning outcomes

After successfully completing this course you should be able to:

1. Implement object-oriented programs according to their specifications.
2. Test components of object-oriented programs (using a testing framework).
3. Write, document and analyse code which uses language features such as inheritance, interfaces, exceptions and I/O.
4. Judge whether a program follows good practice.
5. Write, interpret and critique specifications for program modules (e.g. classes or interfaces).

Assessment

Assessment	Weight	Due	Notes
Ed Lessons / Problem Sets	10%	Weekly Wednesdays 1pm — Weeks 2, 3, 4, 5, 8, 9, 12, 13 (best 6 of 8 count)	Online (Ed Lessons); no extensions (answers released quickly); 100% late penalty
Assignment 1	20%	2/04/2026, 1:00pm	**Hurdle**: the A1 mark is subject to a proportional cap from the A1 Practical — it is the *lower* of your submission score and your practical performance percentage; failing to attend or reschedule the practical makes A1 = 0
Assignment 1 Practical	Pass/Fail	13–17/04/2026, during your assigned practical session	Secure, identity-verified, in-person; contributes to Assignment 1
Assignment 2	25%	15/05/2026, 1:00pm	Implementing/reviewing/debugging/refactoring Java software
Final Exam	45%	End-of-semester exam period, 6/06/2026 – 20/06/2026	**Hurdle**: exam 45% required for Grade 4 and above. Closed book except one double-sided A4 notes sheet; Casio FX82 series or UQ-approved calculator; paper-based, 120 min + 10 min planning

Final mark M (out of 100, rounded before grade cutoffs apply):

$$M = \text{ROUND}(0.1 \times PE + 0.2 \times A1 + 0.25 \times A2 + 0.45 \times E)$$

where PE = Problem sets, $A1$ = Assignment 1 (capped by the A1 Practical), $A2$ = Assignment 2, E = Final Exam.

Grading cutoffs

Grade	Cutoff	Hurdle	
1 (Low Fail)	0–19		
2 (Fail)	20–46		
3 (Marginal Fail)	47–49	Final exam	40%
4 (Pass)	50–64	Final exam	45%
5 (Credit)	65–74	Final exam	60%
6 (Distinction)	75–84	Final exam	70%
7 (High Distinction)	85–100	Final exam	80%

Academic integrity

- All submitted work must be your own (or explicitly-permitted/referenced AI, MT, or third-party code) — checked for plagiarism/collusion, and a subset of students may be interviewed about their submissions to establish genuine authorship.
- Misconduct — things not to do: don't show your code to other students; don't look at other students' code; don't reuse code from other semesters/courses; don't get someone else to write code for you; don't store your code in any online repository others could access (e.g. GitHub, BitBucket) — this counts as showing your code.
- All submitted code is checked for collusion and plagiarism with sophisticated tools; if similarity is found an investigation is launched. **Seek help from course staff in plenty of time** if you get into trouble, and be very careful with outside help — outside tutors may facilitate cheating (and get you caught) rather than your learning.

How to learn (course philosophy)

- Take notes (handwritten recommended), experiment with coding (try compiling/running examples, tweak them and check the effect), ask questions (lectures, Ed forums, practicals), use online resources (Stack Overflow, AI — responsibly).
- Independent learning is emphasised: muddy cards, HW cards, and CodeCamp sessions support self-directed study alongside staff support.