CSSE2002 — Programming in the Large

Table of contents

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 — CSSE2002@eecs.uq.edu.au — office 78-318, consultation via email.

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) — sign up via `signon@eait.uq.edu.au` if you have issues.
- **Ed Lessons** — weekly Java programming exercises (the "Problem Sets" assessment item below).

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**: A1 Practical result caps/reduces this mark (fail practical $\rightarrow$ 50% reduction)
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.

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.