CSSE2002 · PROGRAMMING IN THE LARGE

Exam Sheet Guide

What to put on your one A4 sheet, why it earns the space, and how to lay it out. Ends with a ready-to-print typed sheet, both sides, that you can adapt.

Built for: the current Java 21 course (2026 onward), where the final exam allows one A4 sheet, both sides, typed or handwritten

Inside: the rules for choosing content, a topic-by-topic list of what to include and what to leave off, layout methods, a test routine, and the finished two-sided sheet

Every code template: compiled and run on Java 21, with the printed output being the real output

Written by: a UQ mechatronics engineering graduate who took CSSE2002 and tutored first-year engineering at UQ

Independent study material. Not affiliated with or endorsed by The University of Queensland. All content and code are original. Course details (weightings, exam rules) come from the public course profile and can change, so check the rules for your semester before you print anything.

How to use this guide

The sheet is the only thing you can take into the exam. A good one saves you minutes on every code-writing question and stops the small mistakes that cost marks under pressure. A bad one is a wall of definitions you never look at.

The exam the sheet is for

What	Detail
Weight	55% of the course
Time	120 minutes plus 10 minutes planning
Allowed in	One A4 sheet, both sides, typed or handwritten. Closed book otherwise.
Hurdle	Minimum exam mark for a 3 / 4 / 5 / 6 / 7: 35% / 40% / 55% / 65% / 75%

Because of the hurdle, the exam caps your grade no matter how well the assignments went. Your sheet is part of that exam performance, so it's worth an evening of real work.

Three ways to use this guide

1. **Best:** read sections 1 to 4, then build your own sheet from scratch. Use the sheet at the back only to check you haven't missed anything. Writing it yourself is revision; reading someone else's is not.
2. **Good:** start from the sheet at the back, then cut and rewrite it in your own words, using section 2 to decide what stays. Add whatever your practice papers show you needed.
3. **Last resort:** print the sheet at the back as it is. It will still help, but do at least one timed practice paper with it first, so you know where everything is.

> **CHECK FIRST**
> Exam rules can change between semesters. Before printing, confirm in your course profile and the exam instructions that a typed sheet is still allowed, and whether there are any rules on font, margins or attachments.

Contents

1 What earns a place on the sheet
2 Topic by topic: put on, leave off
3 The code templates, checked
4 Java facts worth a line
5 Layout: making it readable
6 Test your sheet before the exam
7 How to adapt the ready-made sheet
8 The sheet: side 1 and side 2

1 What earns a place on the sheet

Space on one A4 page is the scarcest thing you have. Every line should pass one test: **would I lose time or marks rebuilding this from memory in the exam?**

Five rules

Rule	Why	Example
1. Templates beat definitions	Writing code on paper is slow and error-prone. A correct skeleton for equals, compareTo or a file-reading loop saves minutes and avoids syntax slips. A definition of "encapsulation" saves nothing: you either understand it or you don't.	The 8-line equals and hashCode pattern
2. Procedures beat facts	Reasoning questions (dispatch, substitution, coverage) are lost by skipping a step. A numbered method keeps you on track when you are tired.	"Declared type picks the signature, actual type picks the body"
3. Traps beat basics	You won't forget what a for loop is. You might forget that `-7 / 2` is -3, or that a catch matches subclasses.	A "Java facts" box of the surprising results
4. Tables beat paragraphs	You scan a sheet, you don't read it. A table row is found in two seconds.	Override vs overload in a 5-row table
5. Formats beat content	For test and spec questions, the marks are in the layout of the answer. Having the column headings ready means you start writing straight away.	"Input / expected / covers / why"

What to leave off

- **Anything you could explain to a friend without notes.** If you know it, it's dead space.
- **Long prose definitions.** Replace them with a one-line rule or a picture.
- **Whole programs.** Keep only the reusable shape: the signature and the lines you'd get wrong.
- **Material from other courses**, or library APIs the course never used.
- **Anything copied from slides.** Rewritten in your own words, it sticks. Pasted, you won't recognise it at speed.

Where the space should go

As a method for splitting space (not a rule from the course), aim roughly for this, then adjust after your practice papers:

Share of the sheet	Content
About 40%	Code templates and exact signatures (side 2)
About 35%	Reasoning methods: dispatch, substitution, specs, invariants, testing (side 1)
About 15%	Traps and Java facts
About 10%	Exam routine: planning-time checklist, answer formats