CSSE2002 · PROGRAMMING IN THE LARGE

Hard-Topic Explainers

Deep dives on the three topics that cost the most marks: declared vs actual type with overloading, the substitution principle, and test coverage tables.

Built for: the current Java 21 course (2026 onward)

Each topic: the mental model, worked examples, the traps, a step-by-step answer method and practice questions with answers

Every code example: compiled and run on Java 21, with the printed output being the real output. Lines marked as compile errors were checked against javac too

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 explanations, diagrams and code are original. Course details (weightings, exam rules) come from the public course profile and can change, so check yours.

Why these three topics

Most of CSSE2002 rewards practice. These three topics reward understanding. Students who memorise a rule for them still lose marks, because the exam twists the setup until the memorised rule no longer fits.

Topic	Why it sinks people	What fixes it
1. Declared vs actual type, overloading vs overriding	Two decisions happen at two different times, and students merge them into one. The answers look random until you split them.	A two-step method you apply to every call
2. The substitution principle	"Weaker precondition, stronger postcondition" is easy to recite and easy to apply backwards. Invariants and "never changes" promises get forgotten.	A picture of what each direction means, plus a client test
3. Test coverage tables	Marks sit in the justification column, and in finding the minimum set. Students list tests without showing what each covers.	An outcome checklist and a fixed table format

The exam these feed into

The final exam is worth 55%, runs 120 minutes plus 10 minutes planning, and you may bring one A4 sheet (both sides, typed or handwritten). It also carries a graded hurdle: you need at least 35 / 40 / 55 / 65 / 75% in the exam for a grade of 3 / 4 / 5 / 6 / 7. No compiler, so every question in this booklet is about predicting what javac and the JVM would do.

How to use this booklet

1. **Read the model section of a topic slowly,** once. It is short on purpose.
2. **Do each worked example on paper before reading its answer.** Cover the output box.
3. **Learn the answer method.** Each topic ends with a numbered method for writing the answer. Practise writing it in that order.
4. **Do the practice questions,** then check the answers at the back.
5. **Copy the "sheet lines" box onto your A4 sheet** in your own words.

Contents

1 Declared vs actual type, overloading vs overriding

2 The substitution principle

3 Test coverage tables

1 Declared vs actual type, overloading vs overriding

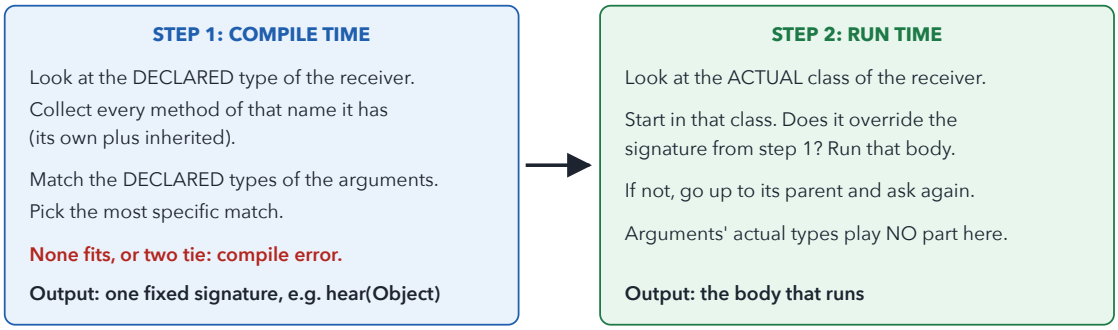
Every method call in Java is decided in two steps. The compiler picks a **signature**. Later, the running program picks a **body** for that signature. Keep the two steps apart and these questions become mechanical.

1.1 Two types for every reference

In `Animal a = new Dog();` the variable `a` has a **declared type** (Animal, what the code says) and refers to an object with an **actual type** (Dog, what `new` made). The declared type never changes. The actual type depends on which object the variable holds at that moment.

- The compiler only knows declared types. It has not run the program.
- The JVM knows actual types, but it can only choose among methods the compiler already approved.

1.2 The two-step model



Two consequences explain nearly every trick question:

- **Overloads are chosen by declared types** (step 1). A better-matching overload in the subclass is invisible through a parent-typed reference.
- **Overrides are chosen by the actual type of the receiver only** (step 2). The actual types of the arguments never matter at run time.

1.3 Overloading vs overriding, side by side

	Overriding	Overloading
Name	Same	Same
Parameter types	**Exactly the same**	**Different** (number, type or order)
Where	Subclass replaces a parent's method	Any class; often the same class
Return type	Same, or a subtype (covariant)	Anything
Decided	Run time, by actual type of receiver	Compile time, by declared types of arguments
`@Override`	Allowed, and catches mistakes	Compile error
Static, private, fields	Never overridden. Static methods and fields are hidden; private methods are not inherited.	

1.4 Worked example: ten calls, one method

Predict all ten lines before reading on. Note that `Parrot` overrides `hear(Object)` but adds `hear(String)` as a new overload.