EECS2030 (B) Advanced Object-Oriented Programming Fall 2026

Digest of Lecture 1, September 9

Course Foundations, Assessment, and Primitive vs. Reference Types

Jackie Wang

How to use this Digest

Use this digest as a **roadmap and checklist**. It highlights the main ideas, important examples, common pitfalls, practice targets, and where to return in the course materials. It does **not** replace attending or reviewing the session, using the recording when needed, studying Jackie's annotated iPad notes and other official course materials, completing assigned work, or practising independently. Use it to identify what you understand, what you need to revisit, and what you should be able to do on your own.

If time is limited, start with the sections most relevant to your current review and return to the rest later; selective use is better than skipping the digest because it looks long.

This is an AI-assisted study roadmap reviewed by Jackie and grounded in the session transcript and Jackie's annotated iPad notes. Like any course document, it may still contain an error or omission. If something seems technically doubtful or inconsistent with the original course materials, return to those materials and ask Jackie for clarification rather than treating the digest as final authority.

Table of Contents

Main Ideas 2

Roadmap 2

Key Terms, Notation, and Structures 4

Key Examples and Demonstrations 5

Common Pitfalls and Misconceptions 6

What You Should Be Able to Do 7

Suggested Practice 7

Where to Review 8

Closing Reminder 9

Main Ideas

- **The course is designed around steady, independent development of OOP skill.** Sections B and E are fully coordinated. The lecture site is the main hub for course materials, while eClass is used for announcements, written tests, and some course-specific resources. Recordings, Jackie's annotated iPad notes, notes templates, and lecture digests support review, but they do not replace active study and independent practice.
- **Labs are low-stress learning opportunities; tests require independent performance.** The lab component is worth 5% in total, and Jackie emphasized submitting an honest best attempt by the deadline. During lab learning you may use support from Jackie, TAs, peers, and generative AI. In tests, by contrast, peer collaboration and those external supports are not allowed; you work individually.
- **The programming-test environment should shape how you practise.** Eclipse is the only IDE allowed in programming tests. Lab0 is therefore a chance to become comfortable with Eclipse, JUnit 4, breakpoints, the debugger, project export, and submission rather than postponing those skills until a test.
- **JUnit starter tests act as precise specifications.** Rather than receiving partially completed class templates, you are expected to infer what classes and methods must be built from starter tests. Jackie's notes connect an API to the public view of classes and methods, including a method's name, return type, and parameter/input types.
- **Foundations matter even more when AI is available.** Jackie emphasized tracing, testing, debugging, and independent reasoning so that you can judge whether a proposed solution really satisfies its requirements. AI can help you learn, but it does not remove your responsibility to understand, test, and revise the software you produce.
- **The first technical review focused on primitive and reference variables.** In Jackie's runtime visualization, an `int` variable stores its value directly, while a variable whose type is a class stores a reference/address to an object elsewhere in memory. The code example also contains an array field, but Lecture 1's runtime explanation focused on `i` and `b1`.

Roadmap

1. Course structure, resources, and expectations

Jackie opened by explaining that Sections B and E are completely coordinated, while students should normally use their enrolled section for scheduled tests because of capacity constraints. Lecture attendance does not directly add or remove marks, but Jackie strongly encouraged in-person learning because the class emphasizes worked reasoning, code tracing, and interaction rather than simply reading slides aloud. For Section B, Jackie can normally remain after class for questions; office hours were announced for Monday and Thursday, 2–3 p.m.

The shared lecture site is the main course hub. Jackie showed the recurring resources associated with each lecture: a notes template, Jackie's annotated iPad notes, a lecture digest, and the lecture recording. eClass remains important for announcements and written-test delivery, so check it regularly. Students who are not yet formally enrolled should keep up with the public early materials and contact Jackie for eClass access; late enrollment is not a reason for an extension.

2. Lab0 and learning from specifications

The first two weeks are intended to establish common ground through Lab0 Part 1 and Part 2. Jackie emphasized following the instructions, videos, and supporting PDFs rather than treating the videos as isolated material. One supporting document shows how to infer Java classes and methods from JUnit test cases. Another focuses on programming patterns for array attributes, which Jackie described as foundational for later labs and programming tests.

From Lab1 onward, the implementation style is deliberately specification-driven: you receive a problem description and starter tests, then develop the required classes from scratch. Jackie highlighted JUnit tests, breakpoints, and the debugger as skills to practise early. Lab submissions also give you practice with the same export/submission flow used for programming tests.

3. Labs, tests, and the grading structure

Labs are worth 5% in total and are primarily for learning. Jackie repeatedly stated that an honest best attempt submitted by the deadline earns the lab credit; students should not withhold a submission merely because the work is incomplete. Collaboration and learning support are acceptable for labs, including peer collaboration and using generative AI as a learning aid. During tests, however, collaboration and external support are not allowed; students must demonstrate the material independently.

There are four programming tests. Programming Test 0 is the first, worth 3%, and is scheduled for September 30 during the enrolled lab session at William Small Centre. The tentative preparation scope mentioned in Lecture 1 was Lab0 Parts 1 and 2 and part of Lab1; Jackie said a later test guide, about a week before the test, would confirm the coverage. He also said he would try to arrange a simpler mock-up opportunity on September 25, with details to be announced. There are two written tests, delivered through eClass in an invigilated lab setting; they are expected to be mostly multiple choice with possible short answers.

The final exam is worth 50%, is cumulative for the technical material beginning with the next lecture, and is a three-hour paper-and-pen exam. Jackie explained that students may need to write small pieces of Java code by hand, such as a method or part of one, but not a full programming-test-sized solution. Minor syntax slips are not the main concern when the intended code is clear and legible.

4. Course learning outcomes and technical direction

Page 2 of Jackie's annotated iPad notes marks CLO1–CLO6 as the active outcomes for this offering and marks CLO7–CLO9 for later courses. The annotations connect API work to the public view of classes and methods and identify three core pieces of a method API: name, return type, and parameter/input types. Testing uses JUnit 4, and Eclipse is the required programming-test IDE.

Aggregation and composition are associated with Lab 3. Inheritance is the major advanced OOP topic later in the term: Jackie previewed both the Java coding mechanisms and the design consequences, including ideas such as polymorphism and dynamic binding. Recursion is planned near the end of the course as a bridge toward EECS2101.

5. Foundations, AI, active study, and professional responsibility

After the syllabus, Jackie used the open letter to explain the philosophy behind the course. The objective is not merely to obtain working code or to recognize familiar practice-question formats. Students should build the ability to trace, test, debug, revise, and explain programs independently. If AI is used while learning, the key questions are whether the prompt reflects the real requirements and whether you can judge and repair the returned solution when it is incomplete or wrong.

Jackie also connected this idea to professional practice and the Professional Engineers Ontario code-of-ethics context: competence and accountability matter when other people may depend on what you build. He recommended seeking help early when study methods are not producing the expected learning, rather than waiting until a major assessment has already gone badly.

6. First OOP review: primitive versus reference variables

The lecture ended with a short runtime-visualization exercise. Given a class with an `int` field, a field whose type is another class, and an array field, Jackie explicitly focused the runtime comparison on `i` and `b1`. He drew the primitive value stored directly in `i` and the class-typed reference in `b1` as an address pointing to a separate object in memory. The array field `b2` appears in the code example, but its runtime treatment was left for the subsequent review material.

Key Terms, Notation, and Structures

Application Programming Interface (API)
: The public view of a programming solution at the level of classes and methods. Jackie's notes connect this directly to what starter tests specify.

Method API
: The method information emphasized in Lecture 1: method name, return type, and parameter/input types. Jackie noted that precise specifications matter before implementation begins.

Starter tests / specification JUnit test cases that specify required classes, methods, constructors, calls, and expected outcomes. In this course they guide development from scratch rather than merely checking code after implementation.

JUnit 4 The unit-testing framework version used in EECS2030. Lab0 introduces the project setup and testing workflow used throughout the course.

Primitive type A type such as `int`, `double`, or `boolean`. In the Lecture 1 runtime picture, a primitive variable stores its value directly.

Reference type In Lecture 1, `b1` is the worked reference-type example. Jackie's runtime picture shows `b1` storing a reference/address to an object elsewhere in memory, not the object's entire contents.

Breakpoint and debugger The tools Jackie wants you to use for step-by-step diagnosis when code does not behave as expected. Lab0 is the place to build this proficiency.

Inheritance The major advanced OOP topic later in the term. Jackie previewed both its Java coding mechanisms and its design consequences, including polymorphism and dynamic binding.

Key Examples and Demonstrations

Reading starter tests as an implementation specification

Setup. Jackie described the recurring lab and programming-test workflow: you are not given half-completed classes and asked merely to fill in blanks. Instead, JUnit starter tests provide a precise specification that your implementation must satisfy.

Main point. Before coding implementation details, inspect the tests to infer the required classes, constructors, method names, parameter types, return types, and expected outcomes. This is the practical connection between the API concept and the programming work in the course.

What to practise. In Lab0, get used to moving from test cases to an implementation plan and then using JUnit results and the debugger to refine the code.

Course-learning-outcome map in Jackie's annotated iPad notes

Page 2 of Jackie's annotated iPad notes is the main visual map of the course. The annotations identify API as the public view of the programming solution, split that view into classes and methods, list method name, return type, and parameter/input type, mark JUnit 4 for testing, note Lab 3 for aggregation/composition, and describe inheritance as involving both coding and design. The same page explicitly marks CLO7–CLO9 for EECS2101/EECS3101 rather than this offering.

Primitive and reference variables at runtime

Page 3 of Jackie's annotated iPad notes contains the key technical diagram from the end of the lecture:

```
class A {
    int i;
    B b1;
    B[] b2;
}
class B { ... }
```

Setup. The class contains the fields `i`, `b1`, and `b2`. Because time was short, Jackie explicitly focused the runtime comparison on `i` and `b1`.

Main point. Jackie drew `i` as a box containing a value such as `23`. For `b1`, he drew a separate memory region for a `B` object and a made-up address/reference stored in `b1`. The important distinction is that `b1` does not contain the entire object itself. The array field `b2` remained part of the source example but was not developed further in this lecture.

What to practise. Redraw the picture and explain, without looking, why `i` and `b1` have different runtime representations. Be ready to extend that reasoning to arrays in the coming review material.

Common Pitfalls and Misconceptions

- **Waiting for a partially completed class template.** The course deliberately expects you to read JUnit starter tests as specifications and develop the required class or classes from scratch.
- **Treating a reference variable as if it contains the whole object.** In the Lecture 1 memory model, the object occupies a separate memory region; the reference variable stores an address/reference used to reach it.
- **Practising only in another IDE and postponing Eclipse.** Other IDEs are acceptable for labs and extra practice, but Eclipse is the only IDE allowed in programming tests. Delaying Eclipse practice creates an avoidable assessment problem.
- **Using `System.out.println` as the main debugging strategy.** Jackie wants you to become comfortable with JUnit tests, breakpoints, and the debugger for systematic, repeatable diagnosis.
- **Using AI to obtain a complete lab solution without developing the underlying skill.** AI is permitted as a learning tool for labs, but it is not available in tests. You need to be able to judge, test, debug, and revise code independently.
- **Assuming a lab must be perfect before submission.** Jackie repeatedly emphasized submitting an honest best attempt by the deadline. The lab is designed as a learning opportunity, not as a reason to risk no submission while chasing perfection.

- **Expecting practice questions to predict the exact test.** Jackie warned that examples and practice material are meant to help you learn the ideas and test workflow, not to be memorized as templates for the actual assessment.
- **Misreading attendance as an ordinary grade component.** iClicker attendance has no direct bonus or penalty. Jackie may use sustained engagement as one indicator when considering borderline final grades, but no adjustment is guaranteed.

What You Should Be Able to Do

After this session, you should be able to:

- locate the main course resources and explain the roles of the lecture site, eClass, recordings, notes templates, Jackie's annotated iPad notes, and lecture digests;
- explain the distinction between low-stress lab learning and independent, invigilated assessment, including why an honest best attempt should be submitted on time;
- explain why Eclipse, JUnit 4, breakpoints, and the debugger need to become familiar tools before the first programming test;
- inspect a starter test and identify the API information it specifies, especially a method's name, return type, and parameter/input types;
- distinguish the primitive variable `i` from the class-typed reference variable `b1` in the comparison developed in class;
- reproduce the basic runtime picture from page 3 of Jackie's annotated iPad notes and explain why `i` stores a value directly while `b1` stores a reference/address;
- explain why tracing, testing, debugging, and independent reasoning remain necessary when AI tools are available; and
- describe the broad course progression from APIs and testing through aggregation and composition, inheritance, and recursion without treating this overview as exact test coverage.

Suggested Practice

- **Read Jackie's open letter during the first week.** Pay particular attention to the sections on foundations in the age of AI, active study, steady preparation, and the role of practice problems.
- **Work through Lab0 Part 1 carefully and continue into Part 2 as it becomes available.** Read the surrounding instructions and supporting PDFs, not only the tutorial videos. Pay special attention to the document on inferring Java classes and methods from JUnit tests and the document on array attributes.

- **Practise the full Eclipse workflow.** Set up JUnit 4, compile and run examples, set breakpoints, step through code with the debugger, export the project, and become familiar with the submission procedure used by the course.
- **Review the prerequisite and background material selectively.** Jackie recommended quickly skimming the optional background slides and using them only where first-year topics such as variables, conditionals, loops, or nested loops no longer feel routine.
- **Redraw the primitive/reference diagram from page 3.** From memory, explain why `i` stores a value directly while `b1` stores a reference/address. Keep `b2` visible as part of the original code example, but leave its runtime analysis for the subsequent review material.
- **Install iClicker and join EECS2030 B/E before the trial checks.** Jackie planned trial attendance checks in the following Monday and Wednesday classes, with ordinary counting beginning the week after. The record has no direct mark bonus or penalty.
- **Keep the first programming-test date visible.** Programming Test 0 is scheduled for September 30. Use the early labs to build the required workflow rather than treating PT0 preparation as a last-minute task.

Where to Review

- **Jackie's annotated iPad notes, page 1:** the Lecture 1 roadmap – syllabus/open letter/introduction, professional-engineering ethics, foundations in the age of AI, and the OOP variable-types exercise.
- **Jackie's annotated iPad notes, page 2:** the course learning outcomes and Jackie's annotations linking APIs to the public view of classes and methods, method API components, JUnit 4, Eclipse, Lab 3, inheritance, and the outcomes reserved for EECS2101/EECS3101.
- **Jackie's annotated iPad notes, page 3:** the code example contains `int i`, `B b1`, and `B[] b2`; the annotated runtime comparison developed in Lecture 1 focuses on `i` versus `b1`. Reproduce that comparison during review.
- **Lecture recording, roughly 0:00–0:38:** course expectations, lecture-site navigation, Lab0 resources, lab policy, test conditions, office hours, and the role of lecture digests.
- **Lecture recording, roughly 0:38–1:01:** course learning outcomes, APIs, JUnit 4, Eclipse, the inheritance/recursion roadmap, grading, final-exam expectations, iClicker, the calendar, and Programming Test 0.
- **Lecture recording, roughly 1:01–1:14:** the open-letter framing, foundations in the age of AI, active study, testing/debugging, regression testing, and professional responsibility.
- **Lecture recording, roughly 1:14–1:17:** the primitive/reference variable exercise corresponding to page 3 of Jackie's annotated iPad notes.

- **Course syllabus, semester calendar, and lab instruction PDFs:** use these as the original source for grading rules, missed-test/accommodation procedures, official deadlines, office hours, test dates, and submission instructions. Programming Test 0 is scheduled for September 30; confirm each lab deadline from the top of its own instruction PDF.
- **Lab0 Part 1 and Part 2 materials:** review the JUnit 4 setup, Eclipse/debugger workflow, starter-test specification guidance, project export and submission procedure, and the highlighted material on arrays used as class attributes.
- **Jackie's open letter and the foundations-in-the-age-of-AI material:** return here for the course's expectations about active learning, independent competence, AI use, steady preparation, and seeking help early.

Closing Reminder

Use this digest as a roadmap for active review, not as a substitute for the learning process itself. Attend Jackie's in-person sessions and pay attention to how he develops ideas, reads specifications, traces code, and works through the important reasoning step by step. If a transition, explanation, policy detail, or technical example remains unclear afterwards, return to the recording to fill that gap.

For structured review, start with Jackie's annotated iPad notes because they most closely reflect the actual flow and development of the class. Work through them in manageable logical units. After each logical unit, study the corresponding part of the presentation slides alongside it. Jackie does not normally teach by reading the slides line by line, so the slides may contain relevant supporting material that was not explicitly read aloud.

Finally, reproduce a suitable selection of the important work from this session on your own: infer API requirements from starter tests, practise JUnit and Eclipse debugging, and redraw the primitive/reference memory picture. The goal is not to memorize Jackie's recorded steps, but to determine whether you can reproduce the reasoning from your own understanding and explain why the important distinctions matter.