EECS2030 (E) Advanced Object-Oriented Programming Fall 2026

Digest of Lecture 14, November 02

`equals`: Phase 3–4, Short-Circuit Guards, and JUnit Equality

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 6

Key Examples and Demonstrations 7

Common Pitfalls and Misconceptions 8

What You Should Be Able to Do 9

Suggested Practice 10

Where to Review 10

Closing Reminder 11

Main Ideas

- **Written Test 1 ends at Phase 3.** Jackie drew the cutoff explicitly: the opening Phase-3 material, through the `p1.equals(s)` trace (approximately 17 minutes into the recording), is in scope. Phase 4 and the later `assertSame/assertEquals` material are outside Written Test 1.
- **Phase 3 rejects objects with different dynamic types.** Reaching Phase 3 already means the objects are distinct and the argument is non-null. The condition `this.getClass() != obj.getClass()` therefore checks whether their dynamic types differ; if they do, `equals` returns false immediately.
- **Phase 4 needs a cast because access follows static type.** Even after Phase 3 establishes that both runtime objects are `PointV2`, parameter `obj` is still declared as `Object`. Direct `obj.x` or `obj.y` therefore does not compile. The cast creates a `PointV2`-typed alias that can access the fields.
- **Equal contents and identical references are different facts.** In the worked `p1.equals(p2)` trace, `p1` and `p2` refer to different objects but their `x`/`y` contents match, so the overridden method returns true.
- **Short-circuit ordering is a safety issue, not just a style choice.** When combining the null check and class check, the safe form begins with `obj == null`. That lets `||` short-circuit before `obj.getClass()` is attempted on a null reference.
- **JUnit separates reference identity from object equality.** Jackie used `assertSame` for reference identity and explained object `assertEquals(exp1, exp2)` through `exp1.equals(exp2)`. The first expression becomes the context object, so argument order can change which `equals` implementation is invoked.

Roadmap

1. Assessment and course priorities

Written Test 1 is Wednesday, November 4 during the in-room lab session and will be delivered through eClass without Eclipse. Jackie told students to bring a phone for Duo Mobile and to verify both their eCS and Passport York logins in advance. He emphasized that tracing is a major preparation priority.

The Written Test 1 guide, practice questions, and assigned exercises are already available. TDD concepts/process from the earlier JUnit/TDD unit remain in scope, as does the assigned short-circuit-evaluation exercise. The cutoff for new content is Phase 3 of `equals`, completed at the beginning of this lecture. Phase 4 and the later assertion material are outside Written Test 1.

Programming Test 2 is Wednesday, November 11 and covers Labs 2 and 3. The Lab2 solution has been released. Lab3 is available, and Jackie said its solution will be released early enough to help with Programming Test 2 review.

2. Quick recap: static type, dynamic type, and casts

Before resuming `equals`, Jackie revisited the previous lecture's model. Static type comes from the declaration and fixes the compile-time expectation of what methods/fields can be accessed. Dynamic type follows the object currently referenced and may change after reassignment.

For a variable such as `o3` declared as `Object` but currently pointing to a `C1` object, `o3.m1()` does not compile because the static type is still `Object`. A cast such as `((C1) o3)` creates an expression with static type `C1`; it does not change the declaration of `o3`.

3. Phase 3 starts from what Phases 1 and 2 already established

Jackie's annotated iPad notes, page 3, visually anchor Phase 3. If execution has reached this phase, then the earlier guards did not return:

- `this == obj` was false, so the two references identify different objects;
- `obj == null` was false, so the argument object exists.

This allows the Phase-3 diagram to begin with two distinct, non-null objects. For a general call `x.equals(y)`, `x` supplies the context object and call by value supplies the parameter reference through the conceptual binding `obj = y`.

4. Phase 3: compare dynamic types with `getClass()`

The Phase-3 guard is:

```
if (this.getClass() != obj.getClass()) { return false; }
```

Jackie described `getClass()` as returning the dynamic type of its context object for this trace. If the dynamic types differ, the two objects are not considered equal by this implementation, so the method returns false before field comparison.

The visual on page 3 emphasizes the two `getClass()` calls on the separate objects and the conclusion that distinct dynamic types lead directly to false.

5. Worked Phase-3 trace: `p1.equals(s)`

Page 4 traces line 9 of the example. `p1` points to a `PointV2` object, while `s` points to a `String` object. Because the context object `p1` has dynamic type `PointV2`, the overridden `PointV2.equals` method is selected.

Inside the method, `this` refers to the `PointV2` object and `obj` receives the reference from `s`. Phase 1 is false, Phase 2 is false, and Phase 3 compares `PointV2` with `String`. The types differ, so the method returns false and Phase 4 is bypassed.

This is the Written Test 1 cutoff Jackie announced in class.

6. Phase 4 begins only after all three guards fail to return

Phase 4 is outside Written Test 1, but Jackie continued the lecture to complete the method. Reaching Phase 4 means:

- `this != obj;`
- `obj != null`; and
- `this.getClass() == obj.getClass()`.

For the current example, the two objects are therefore distinct, non-null `PointV2` objects with the same dynamic type.

7. Why the no-cast field comparison does not compile

Page 5 asks students to consider a hypothetical alternative:

```
return this.x == obj.x && this.y == obj.y;
```

The idea is logically appealing because both runtime objects are `PointV2`. However, `obj` was declared in `equals(Object obj)`, so its static type is `Object`. Jackie stressed that dot notation for fields or methods must respect static type at compile time. `Object` does not expose `x` or `y`, so `obj.x` and `obj.y` are compilation errors.

This is the central reason the cast is necessary: Phase 3's runtime knowledge does not rewrite `obj`'s static type.

8. The cast creates a usable `PointV2` alias

The actual Phase-4 line is:

```
PointV2 other = (PointV2) obj;
```

Jackie first visualized the right-hand cast expression as an anonymous alias to the same object currently referenced by `obj`, but with static type `PointV2`. That reference value is then copied into the named variable `other`, which is also statically `PointV2`.

Because `other` has the correct static type, `other.x` and `other.y` compile. The cast does not change `obj`'s declaration and does not create another `PointV2` object.

9. Worked Phase-4 trace: `p1.equals(p2)`

Page 6 traces line 10. Both context object `p1` and argument `p2` have dynamic type `PointV2`, so the overridden version is selected. Call by value makes `obj` refer to the object referenced by `p2`. The first three guards are all false, so execution reaches the cast.

The cast creates the `PointV2`-typed alias `other`. In the worked example, both objects contain `x = 2` and `y = 3`. Therefore:

```
this.x == other.x && this.y == other.y
```

evaluates to true. Jackie emphasized the distinction: the two references point to different objects, but the overridden method declares them equal because their contents match.

10. Assigned line-11 trace and default-versus-overridden dispatch

Page 7 leaves line 11 as an exercise. Jackie stated that its expected result is false because the contents of `p2` and `p3` do not match exactly; students should reproduce the same static-type, dynamic-type, call-by-value, cast, and field-comparison reasoning independently.

Page 8 then places two same-named `p1` variables from different scopes side by side. One context object has dynamic type `PointV1` and therefore uses the inherited/default `Object.equals`; the other has dynamic type `PointV2` and uses the overridden version. Jackie again stressed that method dispatch depends on the dynamic type of the *context object*, not the argument and not merely the variable name.

11. Equality requirements and null-safe method calls

Page 9 links the phased template to general `equals` requirements. Jackie explicitly connected:

- `x.equals(x)` returning true with Phase 1; and
- `x.equals(null)` returning false with Phase 2.

He also named symmetry and transitivity as properties that depend on the way Phases 3 and 4 are implemented and should be verified through testing.

Jackie then reviewed two ways to avoid a `NullPointerException` when the receiver might be null: an explicit null guard, or short-circuit evaluation such as `x != null && x.equals(...)` so the method call is evaluated only when the receiver is known to be non-null.

12. Combining Phase 2 and Phase 3 with short-circuit evaluation

Page 10 turns the two guards into a four-option reasoning exercise. Let:

- Condition 1 be `obj == null`;
- Condition 2 be `this.getClass() != obj.getClass()`.

Because either condition should cause the method to return false, the operator must be `||`, not `&&`. Order also matters. The safe form is:

```
if (obj == null || this.getClass() != obj.getClass()) {
    return false;
}
```

If `obj` is null, the left side is true and the right side is skipped. Putting the class comparison first is unsafe because `obj.getClass()` would be attempted before the null guard.

13. `assertSame`: reference identity

The assertion material beginning on page 11 is **outside Written Test 1**. Jackie described `assertSame(exp1, exp2)` as passing when the two expressions refer to the same object, corresponding to reference comparison with ==.

In the page-11 diagram, `p3 = p1`, so `p1` and `p3` are aliases and `assertSame(p1, p3)` passes. `p2` refers to a separate object, so `assertSame(p2, p3)` fails.

14. `assertEquals`: primitive values and object references

For primitive arguments, Jackie treated `assertEquals` as a value comparison; the example `i = 10`, `j = 20` therefore fails. A primitive cannot itself be the context object of an `equals` method call.

For reference arguments, page 12 gives the teaching equivalence:

```
assertEquals(exp1, exp2)
    ~ assertTrue(exp1.equals(exp2))
```

The order is important: `exp1` becomes the context object, while `exp2` becomes the argument. The dynamic type of `exp1` therefore determines which `equals` implementation is invoked.

Reversing the assertion arguments is not guaranteed to be equivalent because it changes the context object. If the two expressions have different dynamic types or different overrides, the two calls can execute different `equals` implementations.

15. Where Lecture 14 stops

Jackie stopped after establishing the `assertEquals` context-object rule. For Wednesday, he said only a few additional tracing examples and equality with array attributes remain before the course moves into inheritance.

He also recommended spending about 5–10 minutes on the **Student Management System** design exercise under the Inheritance section, first thinking about how the design might be approached without inheritance.

Key Terms, Notation, and Structures

Phase 3 of overridden `equals`
`if (this.getClass() != obj.getClass()) { return false; }`. It rejects two non-null, distinct objects whose dynamic types differ.

Phase 4 of overridden `equals`
The field-comparison phase reached only after identity, null, and class mismatch checks have all failed to return.

Static type
The declared type of a reference expression/variable. Jackie uses it to determine the compile-time expectation for accessible methods and fields.

Dynamic type
The runtime type of the object currently referenced. Jackie uses the context object's dynamic type to choose which `equals` implementation is invoked.

Cast expression
An expression such as `(PointV2) obj` whose static type is the cast type. In Jackie's diagram, it is an alias to the same object rather than a new object or a permanent change to `obj`.

Reference-equal versus contents-equal
Reference-equal means two references identify the same object. Contents-equal in the current `PointV2` override means the compared fields satisfy the method's field-comparison condition.

Short-circuit evaluation
Left-to-right boolean evaluation that can skip the right operand when the result is already determined. Here it protects `obj.getClass()` from being called when `obj` is null.

`assertSame`
A JUnit assertion used here for reference identity: the two evaluated references must identify the same object.

`assertEquals`
For object references in this lecture, Jackie explains it through the first argument becoming the context object of an `equals` call on the second argument.

Key Examples and Demonstrations

Phase 3: `p1.equals(s)`

Setup. Page 4 shows `p1` with dynamic type `PointV2` and `s` with dynamic type `String`.

Main point. The context object selects the overridden `PointV2.equals`. After Phase 1 and Phase 2 both evaluate false, Phase 3 compares the two dynamic types, sees `PointV2 != String`, and returns false.

What to practise. Reconstruct `this`, `obj`, and the three guard results without copying Jackie's finished arrows. This trace marks the Written Test 1 cutoff.

Phase 4: why the cast is necessary

Setup. Page 5 first removes the cast and proposes direct `obj.x`/`obj.y` access.

Main point. The attempt fails at compile time because `obj`'s static type is `Object`. The actual runtime object being `PointV2` does not change the compiler's view of `obj`.

What to practise. For every dot expression, identify the static type of the expression to the left of the dot before deciding whether the field or method is available.

Phase 4 trace: different references, equal contents

Setup. Page 6 traces `p1.equals(p2)` with two different `PointV2` objects whose fields are both `(2, 3)`.

Main point. `other` is a `PointV2`-typed alias to the argument object. Both field comparisons are true, so the overridden method returns true even though `p1` and `p2` are different references.

What to practise. Keep separate annotations for object identity, static type, dynamic type, and field values.

Short-circuit guard: choose the safe ordering

Setup. Page 10 combines `obj == null` with `this.getClass() != obj.getClass()`.

Main point. The correct classroom option is the null check first with `||`. That ordering can short-circuit before `obj.getClass()` is called on a null reference.

What to practise. Evaluate each candidate left to right and state exactly when the right operand is or is not evaluated.

JUnit identity and equality

Setup. Page 11 shows `p3 = p1`; page 12 rewrites object `assertEquals` through an `equals` method call.

Main point. `assertSame` follows reference identity. For object `assertEquals(exp1, exp2)`, `exp1` becomes the context object of `exp1.equals(exp2)`.

What to practise. For reversed `assertEquals` arguments, identify the new context object and its dynamic type before predicting which `equals` implementation is called.

Common Pitfalls and Misconceptions

- **Studying Phase 4 as part of Written Test 1.** Jackie explicitly ended Written Test 1 coverage after the Phase-3 trace. Phase 4 is later material.
- **Forgetting the logical information carried into a later phase.** Reaching Phase 3 already means Phase 1 and Phase 2 did not return; reaching Phase 4 means all three earlier guards did not return.
- **Using dynamic type to justify direct field access through `obj`.** Field/method access must still satisfy the static type. `obj` remains statically `Object`.
- **Believing a cast changes `obj`'s declared type or creates a new object.** Jackie models the cast as an alias expression to the same object with a different static type.
- **Confusing equal contents with reference identity.** The Phase-4 example returns true for two distinct objects because their fields match.
- **Choosing an `equals` implementation from the argument.** Dispatch follows the dynamic type of the context object.

- **Calling `obj.getClass()` before checking whether `obj` is null.** In the combined guard, that ordering can produce a `NullPointerException` before the null check is reached.
- **Assuming `assertEquals(exp1, exp2)` and `assertEquals(exp2, exp1)` must behave identically.** Reversing the arguments changes the context object and can therefore change the invoked `equals` implementation.
- **Treating same-named variables in different scopes as the same variable.** Jackie's side-by-side `PointV1`/`PointV2` example shows that repeated names can denote different references with different dynamic types.

What You Should Be Able to Do

After this session, you should be able to:

- state the exact Lecture-14 cutoff for Written Test 1 and distinguish it from material taught later in the same lecture;
- explain what reaching Phase 3 implies about `this` and `obj`;
- trace `this.getClass()` and `obj.getClass()` and use the result to decide whether Phase 3 returns false;
- reproduce the `p1.equals(s)` trace through Phases 1–3;
- state the three facts known when execution reaches Phase 4;
- explain why direct `obj.x`/`obj.y` access does not compile even when `obj` dynamically references a `PointV2`;
- explain what `(PointV2) obj` changes and what it does not change;
- trace the creation of `other` and the final field comparisons in `p1.equals(p2)`;
- distinguish reference identity from matching object contents;
- identify whether the inherited/default or overridden `equals` version is selected from the context object's dynamic type;
- explain why `x.equals(x)` maps to Phase 1 and `x.equals(null)` maps to Phase 2 in Jackie's discussion;
- choose the safe short-circuit ordering for combining the null and class-mismatch guards and justify it left to right;
- determine whether a simple `assertSame` assertion passes from a reference diagram; and
- translate object `assertEquals(exp1, exp2)` into the corresponding `equals` call used in this lecture and identify the context object.

Suggested Practice

- For **Written Test 1 on Wednesday, November 4**, review the Phase-3 `p1.equals(s)` trace, the earlier static/dynamic-type and casting example, the assigned short-circuit-evaluation exercise, and the earlier TDD process/concepts Jackie identified as testable.
- Rehearse tracing without Eclipse. Jackie said tracing is a major Written Test 1 focus, and the test itself will be completed on eClass without Eclipse access.
- Redraw page 3 of Jackie's annotated iPad notes: two distinct non-null objects, `this`, `obj`, and both `getClass()` results. Then redo the page-4 `p1.equals(s)` trace independently.
- For post-test review, work through page 5's no-cast alternative and explain exactly why `obj.x` and `obj.y` are outside the expectation of static type `Object`.
- Redo the page-6 `p1.equals(p2)` trace and then complete the assigned page-7 line-11 exercise, whose expected result is false.
- Revisit page 10's four short-circuit choices. Evaluate each expression left to right and identify where a null dereference can occur.
- Reconstruct the page-11 reference diagram for:
  `assertSame(p1, p3)` and `assertSame(p2, p3)`.
- For the page-12 `assertEquals` rule, swap the two reference arguments and determine which expression becomes the context object in each order.
- Begin Programming Test 2 preparation by reviewing the released Lab2 solution and continuing Lab3. Jackie said Programming Test 2 on November 11 covers both labs.
- Before Wednesday's inheritance transition, spend about 5–10 minutes on the **Student Management System** design exercise and think about a design without using inheritance.

Where to Review

- **Jackie's annotated iPad notes, page 2:** Written Test 1 and Programming Test 2 reminders, released preparation resources, Lab2/Lab3 priorities, and the Student Management System design exercise.
- **Jackie's annotated iPad notes, pages 3–4:** Phase-3 setup, the two `getClass()` calls, and the complete `p1.equals(s)` trace that marks the Written Test 1 cutoff.
- **Jackie's annotated iPad notes, pages 5–6:** Phase 4, the no-cast compilation problem, `PointV2 other = (PointV2) obj`, and the worked `p1.equals(p2)` trace where different objects have equal contents.
- **Jackie's annotated iPad notes, pages 7–8:** the assigned line-11 trace and the side-by-side default-versus-overridden `equals` dispatch across different scopes.

- **Jackie's annotated iPad notes, pages 9–10:** the first two stated `equals` requirements, null-safety guidance, symmetry and transitivity, and the four-way short-circuit exercise combining Phases 2 and 3.
- **Jackie's annotated iPad notes, pages 11–12:**
  `assertSame` reference identity, primitive `assertEquals`, and object `assertEquals(exp1, exp2)` rewritten through `exp1.equals(exp2)`.
- **Lecture recording, roughly 0:00–0:17:**
  Assessment scope, Programming Test 2/Lab reminders, TDD clarification, static/dynamic-type and cast recap, Phase 3, and the `p1.equals(s)` Written Test 1 cutoff trace.
- **Recording, roughly 0:18–0:41:** Phase-4 preconditions, no-cast compilation failure, cast-as-alias reasoning, `p1.equals(p2)`, the assigned line-11 exercise, and default versus overridden dispatch across scopes.
- **Recording: roughly 0:41–0:56**
  `equals` requirements, null-safety patterns, symmetry and transitivity, and the combined short-circuit guard exercise.
- **Recording: roughly 0:56–1:08**
  `assertSame`, primitive and reference `assertEquals`, the context-object rule, and why reversed reference arguments need not invoke the same `equals` implementation.

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 and works through the important reasoning step by step. If a transition, explanation, example, or reasoning step 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, independently reproduce a suitable selection of this session's key reasoning: trace Phase 3 and the Written Test 1 cutoff example; explain from static type why the no-cast Phase-4 alternative fails; redraw the cast and `other` alias in the field-comparison trace; evaluate the combined null and class guard left to right; and reconstruct the `assertSame` and `assertEquals` reference reasoning. The goal is not to memorize Jackie's recorded arrows or outcomes, but to determine whether you can reproduce the reference, type, dispatch, short-circuit, and equality reasoning from your own understanding.