Lab 06: The Confusion

You Just Got Challenged

If you're reading this, you probably just experienced confusion, frustration, or both while working on Lab 06. Maybe you created 6 separate files and got errors. Maybe you asked me "which is correct?" Maybe you're still not sure what happened.

Good news: The confusion was intentional.

Let me explain why.

What You Saw

Lab06.java Starter Code Said:

```
// TODO: Create a separate file called Course.java with:
//       - 2 public properties: department (String), number (int)
//       - A print() method that prints department and number together

// TODO: Create a separate file called Student.java with:
//       - 3 public properties: name (String), year (int), gpa (double)

// TODO: Create a separate file called PlayerCharacter.java with:
// ... and so on for all 5 classes
```

Lab06.md Instructions Said:

- **Task 1:** "Create a class called Course (You need to create a file called Course.java)"
- **Tasks 2-5:** Just said "Create a class called..." (no mention of separate files!)

Your Lecture Notes Said:

- "Only ONE public class per file"
- "The public class name must match the filename"
- "Helper classes can be non-public in the same file"

The Boilerplate Had:

- Everything in ONE file (Lab06.java)

The Contradiction

Here's what was contradicting what:

Source	What It Told You	Number of Files

Source	What It Told You	Number of Files
Lab06.md	Only Task 1 mentioned separate file	1 or 2? Unclear.
Lab06.java TODO comments	ALL tasks said "create separate file"	6 files total
Lecture slides	"One public class per file" rule	Could work either way
Boilerplate	Everything in Lab06.java	1 file only

Why I Did This

Reason #1: Comments Lie (In Real Life)

In the real world, code gets updated but comments don't. You'll encounter:

```
// TODO: This function uses recursion
// (Written in 2015, changed to iteration in 2023, but comment never
updated)
public int calculate(int n) {
    // This is actually iterative now, not recursive!
    int result = 0;
    for (int i = 0; i < n; i++) {
        result += i;
    }
    return result;
}
```

The comment says one thing. The code does another. Which do you trust?

Reason #2: Documentation Conflicts (All The Time)

You'll join a company and see:

```
README.md says: "Run npm start"
package.json says: "npm run dev"
Wiki says: "npm run serve"
Your coworker says: "Just use yarn"
```

Which is correct? You have to figure it out.

Reason #3: I'm Testing If You Think Critically

This lab had two tests:

Surface Objective: Can you write Java classes with properties and methods?
Hidden Objective: Do you follow TODO comments, or do you question conflicting information? How do you

handle situations like this?

The Real Answer to "One File or Multiple?"

For Lab 06 specifically: One file is better.

Why?

- Classes are simple (< 30 lines each)
- No reusability needed
- Focus should be on learning OOP, not file management
- Easier to debug when everything is visible

What You Learned In addition to Java Knowledge (hopefully)

If You Got Confused and Asked Questions:

You learned:

- ✅ To notice when sources contradict each other
- ✅ To ask for clarification when uncertain
- ✅ That questioning instructions is smart, not rude
- ✅ That professors are humans who expect you to think

If You Tried Multiple Files First:

You learned:

- ✅ Error messages teach you things
- ✅ Why one file vs multiple files matters
- ✅ The rules about public vs non-public classes
- ✅ Testing and failing is part of the process
- ✅ Persistence matters more than getting it right immediately
- ✅ Debugging is a skill you develop over time

If You Researched On Your Own:

You learned:

- ✅ Googling "java multiple classes one file" solves problems
- ✅ Official documentation trumps random comments
- ✅ Self-directed learning is a superpower
- ✅ You can figure things out without hand-holding

If You Followed TODOs and Got Stuck:

You learned(hopefully):

- ⚠ Comments aren't always correct
- ⚠ You need to verify instructions
- ⚠ When stuck, ask for help sooner
- ⚠ Problem solving & Critical thinking

If You Just Made It Work Without Reflecting:

You learned(hopefully):

- ⚠ The importance of understanding the "why" behind your code
- ⚠ That quick fixes might lead to bigger problems later
- ⚠ The value of taking time to think through a problem
- ⚠ Reflection is key to becoming a better developer

This Happens in Real Jobs

Scenario 1: Legacy Codebase

You'll inherit code that looks like this:

```
// TODO: Delete this function after Q1 2020 migration
// (It's now 2025. Function is still critical. Comment is wrong.)
public void processPayment() {
    // If you delete this, the entire payment system breaks
}
```

Do you trust the comment and delete it? Or investigate first?

Scenario 2: Outdated Stack Overflow

You'll search for solutions and find:

```
Top Answer (2012): "Use jQuery for everything!"
Your Project (2025): Built with React
Comments: "This is the best approach!"
```

Do you copy-paste 2012 code into your 2025 React app? Or adapt?

Scenario 3: Contradicting Coworkers

You'll ask for help and get:

- **Senior Dev:** "Always use multiple files for organization"
- **Tech Lead:** "Keep it simple, one file is fine for small projects"
- **Your Manager:** "Just make it work by Friday"

All three are right in different contexts. You have to decide.

How Different Students Reacted

The Question-Askers ✅

What they said:
"Professor, the TODO comments say separate files, but lecture said one public class per file. Which should I do?"

What they demonstrated:

- Critical thinking
- Willingness to ask for help
- Ability to notice contradictions

Result: Learned the answer quickly and moved on.

The Researchers ✅

What they did:
Googled "java multiple classes in one file", read Oracle documentation, implemented correctly

What they demonstrated:

- Self-directed learning
- Research skills
- Independence

Result: Solved it themselves and built confidence.

The Trial-and-Error Testers ✅

What they did:
Created 6 files → got errors → debugged → tried 1 file → success

What they demonstrated:

- Persistence
- Learning from mistakes
- Scientific method (hypothesis, test, revise)

Result: Understand the "why" deeply now.

The Doers (Just made it work Crowd) ❌ → ✅

What they did:
Created 6 files because TODO said so → got stuck → made multiple files work or eventually asked for help

What they learned(hopefully):

- Comments can be wrong
- Verify before following
- Is your solution optimal?

Result: Learned a perhaps different lesson.

The Answer Hunters ⚠

What they did:
Im thinking probably asked AI or searched copied whatever they found without proper verifying.

What they missed:

- The struggle that builds problem-solving skills
- The chance to develop critical thinking
- The meta-lesson about questioning authority

Result: Code may or may not work, missed a learning opportunity tho.

The Real Lessons

What This Lab Was Really About

You were NOT only learning: How to create Java classes

You were ALSO learning: How to think like a java developer or professional developer in general

---

The Skills You Developed

Technical Skills:

- ✅ Java class syntax
- ✅ Public vs non-public classes
- ✅ File organization rules (I will cover this again later)
- ✅ When to use one file vs many

Professional Skills:

- ✅ Critical thinking
- ✅ Questioning inconsistencies
- ✅ Verifying information across sources
- ✅ Asking clarifying questions
- ✅ Debugging with patience
- ✅ Handling ambiguity

The professional skills matter more in the long run.

What This Means Going Forward

In Your Next Labs

Your job:

- Use the tools and knowledge you have learned in this class, solve the problems with use case, development context, and best practices in mind.
- Question and verify
    - Sometimes this whole process you will do on your own including answer your own questions with either research or trial and error or using slides etc.

Welcome to Software Development (Real World Context)

Where:

- Comments lie
- Documentation conflicts
- Requirements change
- "Best practices" depend on context
- You have to think for yourself

You're not just learning to code.
You're learning to think like a java developer.

Conclusion

To The Question-Askers

Thank you for asking. That's exactly what professionals do. Keep doing it.

To The Researchers

Thank you for Googling. Self-directed learning is how you'll grow your entire career. Keep doing it.

To The Trial-and-Error Testers

Thank you for persisting. Debugging is 80% of the job. Keep doing it.

To The "Made It Work" Crowd

Thank you for your effort. Sometimes getting it to work is the first step. Just remember to reflect on the process and learn from it.

To Everyone Who Got Confused

Perfect. Confusion means you're learning. Lean into it.