lectures # week-1

Week 1: Course Introduction (Monday, January 5, 2026)

Lecture Recording

Watch on Panopto

Related Assignment

This lecture introduces Assignment A1a.

Course Philosophy: The HOW vs the WHY [1:44]

TCSS 305 builds on what you learned in TCSS 142 and 143. Those courses taught you **how** to program—syntax, classes, methods. This course focuses on **why** we do things the way we do.

Think of this as "Programming 3" capping off the programming sequence, but also as "Software Development 0"—your real introduction to professional software development practices.

Key concepts: encapsulation, access modifiers, software development lifecycle

Example: Why do we use `private` for instance fields?

In 143, the answer might have been "because it's on the test" or "encapsulation." But what does that actually mean? The real answer: when your code goes into production, it lives for years and many people work on it. Access modifiers protect the integrity of your design and make code maintainable.

> **The Real World Isn't Friday Submissions**
>
> In industry, code doesn't end when you submit it to your professor. It goes through code review, gets deployed, maintained, and modified by teams over years. That's why we care about access modifiers, documentation, and clean design.

Course Logistics [6:05]

Platforms:

- **Canvas** - Course administration, grades, weekly modules
- **Course Materials Site** - Assignments, guides, readings (built with Claude Code)
- **Discord** - Questions, quick help, office hour scheduling
- **GitHub Classroom** - Assignment submission (new this quarter)

Communication:

- **Email** - Privacy-related matters only (FERPA protected)
- **Discord** - Everything else (not FERPA compliant, so no grade discussions publicly)

> **Use Your Real Name on Discord**
>
> The instructor needs to know who you are. No "Commander PoopyPants 55" usernames.

Course Materials and Gen AI Philosophy [16:58]

The course uses a custom materials site built with Claude Code over 5 days—no expensive textbooks required. This site contains all assignments, guides, and readings.

On Generative AI:

This course leans *into* what LLMs can do, not away from them. However, there's a critical distinction:

- **Gen AI is excellent at writing simple programs** - It can complete A1a, A1b, A1c, A2, and A3 with the starter code and assignment docs
- **What you learn by having Gen AI do your work:** Gen AI is really good at simple programs. That's it.

- **What Gen AI still needs:** An architect. A manager. The vision to go from A to B to C to D.

The amount of code you'll write in your career is going *down*, not up. That makes understanding the WHY even more critical—you need to be able to architect solutions and direct AI tools effectively.

> **Key Insight**
>
> "I truly believe for you to be able to get there, you need to learn to program. You need to learn the WHY of what we're doing here."

Technology Stack [20:22]

Language: Java JDK 25

You're in Computer Science, not "Java Programming." The language is a tool. When you start a job, you ask "What's the tech stack?" and adapt—not insist on your preferred language.

This course teaches modern Java (JDK 21-25 features), not how Java was written 15-25 years ago. Expect expression-based programming patterns.

IDE: IntelliJ IDEA

- Every assignment is an IntelliJ starter project
- Every submission must be a working IntelliJ project
- You can use other tools during development, but it must run in IntelliJ when graded

> ⚠ **Submission Format**
>
> If your project doesn't work in IntelliJ, it's an easy zero. The instructor won't spend time converting your VS Code or Eclipse project.

Mindset: Don't get attached to tools. The more IDEs you experience, the easier it is to learn whatever weird proprietary IDE your future job uses.

Grading Structure [28:35]

Component	Weight
Assignments (A1a, A1b, A1c, A2, A3)	40%
Group Project (weeks 7-10)	30%
In-class Tests (3 total, handwritten)	30%

Assignments:

- Work alone, no partners
- Due Sunday at 11:59 PM (but see late policy below)

Late Policy:

- Up to 2 days late with 10% penalty per day
- "Free extension hack": Due date is Sunday → one day late = Saturday, two days late = Friday
- Think of it as: your assignment is due Friday with a free 2-day extension

Group Project (Weeks 7-10):

- Groups of 4 (assigned by instructor based on prior assessments)
- Build the front-end for a game (likely Tetris)
- Expect some pain points—different motivation levels on teams

Tests:

- In-person, handwritten, pencil and paper
- One page of handwritten notes allowed (front and back)
- Test percentage is 30%—lower than many colleagues who are responding to Gen AI by making tests 60%

Canvas and Materials Site Structure [43:05]

Canvas Modules:

- Weekly modules with topics, readings, deliverables
- Assignments listed but NOT submitted through Canvas

Materials Site Navigation:

- Assignment-based navigation across the top
- Each assignment has an instructions page and guides page
- Guides are the "textbook"—read them!

> **Actually Read the Assignment Document**
>
> Everything you need to do is in there. Take the time to read it. The callouts, hints, and notes exist for a reason.

Recommended book (optional): *Effective Java* by Joshua Bloch—"best programming book I've ever read"

GitHub Classroom [53:42]

New submission system this quarter (first time using it).

Setup Process:

1. Accept the assignment via the GitHub Classroom link
2. Link your GitHub username to your Canvas name (pick the right one!)
3. GitHub creates a repo from the template
4. Clone and work locally
5. Commit and push to submit

Submission:

- Work on the `main` branch
- Commit = save to local repo
- Push = submit to remote (GitHub)
- Last commit timestamp before due date determines if it's late

> **Pick the Right Name**
>
> If you accidentally pick the wrong name in GitHub Classroom, tell the instructor immediately. If you do it on purpose... don't.

Assignment 1a Overview [1:01:34]

A1a is your environment setup and Java review assignment.

Starter Code Structure:

- `WRONG` package - Contains `CheckstyleRuleBreaker` with intentional style violations
- Purpose: Verify your linting tools are installed correctly
- Delete this package before submission
- `StarterApplication` - Fix Checkstyle warnings, implement Hello World messages
- `Item` interface - You implement a class that follows this contract

Key Concept: Interfaces as Contracts

An interface defines WHAT, not HOW. It's a contract between teams.

Analogy: Team in New York implements the interface. Team in San Francisco uses it. The interface guarantees `getName()` returns a `String`—the SF team doesn't need to know if NY stores the name in a database, gets it from an API, or keeps it in memory.

```
public interface Item {
    String getName();
    // The implementing class MUST provide this method
    // Callers know they'll get a String back
    // How it's implemented is an internal detail
}
```

Four Pillars of OOP for A1a:

- Inheritance - Not needed here
- Polymorphism - Not needed here
- Abstraction - The interface IS the abstraction
- **Encapsulation** - This is your focus. Private instance fields, controlled access.

Unit Tests Provided:

- Test file in the `test` folder
- Run tests as you implement—watch them turn green
- A1b will have YOU writing tests

Key Takeaways

1. **This course is about WHY, not just HOW** - You learned syntax in 142/143; now learn the reasoning behind professional practices
2. **Gen AI changes things** - Code volume is decreasing; architectural understanding is increasing in value
3. **Tools are just tools** - Java, IntelliJ, whatever—be adaptable
4. **Read the assignments** - Everything you need is documented
5. **Interfaces are contracts** - They define WHAT, not HOW
6. **Encapsulation matters** - Code lives beyond your submission date

---

This lecture outline is part of TCSS 305 Programming Practicum, School of Engineering and Technology, University of Washington Tacoma.