Class

1

CSC 240 Computer Science III

Introduction

Dr. Si Chen (schen@wcupa.edu)

Tue / Thu · 9:30 – 10:45 AM · Anderson Hall 315
Fall 2026 · Week 1 · Tuesday, August 25, 2026
Prerequisite: CSC 142 · wcu.ninja/csc240

PART 1

Where CSC 240 fits

You can already make code work. This course is about making it correct, reusable, maintainable.

The one-sentence version

- CSC 240 focuses on **advanced object-oriented programming** — plus project design, planning, and testing with milestones and checklists.
- Core topics: **text processing**, **inheritance & polymorphism**, **abstract classes & interfaces**, **exceptions**, **generics**, **linked lists**.
  - Language: **Java**. IDE: **IntelliJ IDEA** recommended (Eclipse/NetBeans/jGRASP accepted).
- Textbook: Gaddis, *Starting Out with Java: From Control Structures through Data Structures*.
- The theme: not "more syntax," but **design that survives contact with change**.

Grade breakdown

4 programming assignments — 30%

2 short quizzes — 10%

Midterm (Ch. 9–10) — 20%

Final (comprehensive) — 30%

Attendance / participation — 10%

HW3 is double-weighted (12%)

What you already know (CSC 141 / 142)

Programming basics

- Java fundamentals; variables, types, operators
- Decision structures (`if` / `switch`)
- Loops and files (`while`, `for`)
- Methods — parameters, return values, scope

Objects & structures

- A first & second look at **classes and objects**
- Arrays and the **ArrayList** class
- Recursion
- A first look at GUI applications

What you will be able to do by December

- Plan, implement, test, and refine OO Java projects using **milestones, checklists, and systematic debugging**.
- Process text with `String`, `StringBuilder`, `StringTokenizer`, and wrapper classes.
- Design **class hierarchies**: inheritance, overriding, polymorphism, abstract classes, interfaces.
- Write **generic** methods and classes for type-safe, reusable code.
- Define, throw, catch, and propagate **exceptions**, including custom exception classes.
- Use advanced **file I/O** and implement **linked-list** operations.

PART 2

Why design is the skill that survives AI

AI writes syntactically perfect Java in seconds. So what are humans for?

The ground shifted while you were in 142

20–30%

of code at Microsoft is now AI-generated — "fantastic" at some languages

Nadella, LlamaCon 2026

75%

of new code at Google is AI-generated and human-approved

Pichai, Google Cloud Next 2026

Reviewer

the entry-level engineer's job is shifting from author to reviewer of AI output

Google, DevOps.com 2026

If AI drafts most of the code, the human job becomes: design the right structure, spot the wrong ones, and be accountable for correctness. That is exactly what CSC 240 teaches.

AI is great at syntax and terrible at judgment

The security gap

- AI-generated code passes basic **security checks only ~56%** of the time — flat year-over-year.
- It produces **compilable** code ~100% of the time — looking right ≠ being right.
- Cross-site scripting: **15%** pass. Log injection: **12%** pass.
- A Fortune-50 study: AI-assisted devs ship **3–4× more commits** — but **10× more security findings**.

The productivity illusion

- A **randomized trial (METR)** found experienced devs were **19% slower** with AI —
- ...yet **believed** they were **20% faster**. Confidence ≠ competence.
- The gap: AI is fluent, not correct. **Someone has to know the difference.**
- In 240 you become that someone — by learning what *correct design* means.

> *Do not treat inheritance, polymorphism, exceptions, generics, and linked structures as isolated syntax. Draw object relationships and trace runtime behavior before coding.*

— CSC 240 Syllabus — "How to Succeed"

How we use AI in CSC 240

Encouraged

Conceptual explanations of Java APIs.

Interpreting compiler / runtime errors.

Small illustrative examples.

General debugging strategies.

Not allowed on graded work

Generating substantial parts of the required solution and submitting as your own.

Unless an assignment explicitly authorizes it.

Always your responsibility

Explain, trace, or modify any code you submit — on request.

No AI on quizzes/exams unless authorized.

Design decisions are yours to defend.

PART 3

Two ways to structure a program

The mental shift at the center of CSC 240: from procedures to objects

Procedural programming

- Older languages (C, Fortran) are **procedural**.
- A **procedure** is a set of statements that together perform a specific task.
- Procedures **operate on data that is separate from them** — data is passed procedure to procedure.
- Data tends to be **global** to the program.
  - **Problem:** when the data format changes, every procedure that touches it must change too. Fragile at scale.

Object-oriented programming

The idea

- OOP centers on **objects**, not procedures.
- An object **melds data and the procedures that manipulate it**.
- Data in an object = **attributes** (fields).
- Procedures in an object = **methods**.

In Java

```
`class Dog {
  private String name;   // attribute
  public String speak(){ // method
    return name + " woofs"; }
}

// each Dog object bundles its own
// data + behavior together
```

Encapsulation & data hiding

- OOP combines data and behavior via **encapsulation** — bundling them in one object.
- **Data hiding:** an object can **hide its attributes** from other objects (`private`).
- Only an object's **own methods** should directly touch its attributes.
- Other objects interact **only through the methods** — the object's **public interface**.
  - Benefit: you can change the inside without breaking everyone who uses the outside. This is *maintainability*.

Procedural vs. object-oriented — side by side

Procedural

- Data is **global / passed around**.
- Functions and data are **separate**.
- A data-format change **ripples everywhere**.
- Good for small, linear tasks.

Object-oriented

- Data is **owned by objects** (private).
- Data and behavior are **bundled**.
- Change the inside; the **interface stays**.
- Scales to large, evolving systems.

PART 4

The road ahead

Five chapters that turn you from a coder into a designer

Course content for CSC 240

Ch. 9 — Text Processing

String, StringBuilder, StringTokenizer.

Wrapper classes for primitives.

Our first topic (Week 2).

Ch. 10 — Inheritance

Superclass/subclass, overriding.

Polymorphism, abstract classes, interfaces.

The heart of OOP design.

Ch. 11 — Exceptions & I/O

throw / try / catch / finally.

Custom exceptions.

Random-access files, serialization.

Ch. 18 — Generics

Type-safe, reusable methods & classes.

Type parameters <T>.

Why ArrayList<String> works.

Ch. 20 — Linked Lists

Nodes, references, traversal.

Insertion / removal.

Your first real data structure.

Throughout — Discipline

Milestones & checklists.

Systematic testing & debugging.

Design before you type.

PART 5

Diagnostic: your 141/142 foundations

A no-stakes check of the skills we build on — you can also take it online at wcu.ninja/csc240/diagnostic

How this works

- This is a **diagnostic, not a quiz** — nothing here is graded.
- It covers **CSC 141/142 foundations** — loops, arrays, and class design.
- Answer on paper or in your head; we work through each together.
- Prefer to submit online (name + student ID)? **wcu.ninja/csc240/diagnostic** — I see your results in the backend.
  - Keep score privately: how many can you answer confidently before we reveal the answer?

Q1 · Gauss Sum (CSC 141, loops)


```
int start = 81297;
int end   = 100899;
int step  = ____;   // ?

long sum = 0;
for (int v = start;
         v <= end;
         v += step) {
    sum += v;
}
System.out.println(sum);
```

- Gauss’s teacher’s problem: **81297 + 81495 + 81693 + … + 100899**, where each step adds the same amount.
  - What is the **common step** between terms?
  - **How many terms** are there?
  - Write a **`for` loop** that accumulates the sum.

Q2 · Array processing (CSC 142, arrays)

DIAGNOSTIC Q2 ⏱ 5 min

```
double sum = 0, max = a[0];
int maxIdx = 0;
for (int i = 0; i < a.length; i++) {
    sum += a[i];
    if (a[i] > max) {
        max = a[i];
        maxIdx = ____;   // ?
    }
}
// avg? min? even-index sum?
```

- For `double[] a = {4.0, 9.0, 7.0, 2.0, 9.0, 5.0};` give each value:
  - **Sum**, **average**, **max**, **min**.
  - The **index** of the max (and of the min).
  - Sum of the **even-index** elements (index 0, 2, 4, …).
- **Discuss:** why track the *index*, not just the max value?

Q3 · Design a class — the bridge to 240


```
public class Bulbasaur {
    private int id = 1;
    private int level = 1;
    public void setLevel(int lv){
        level = lv;
        if (lv >= 32)      id = 3;
        else if (lv >= 16) id = 2;
    }
    public String getName(){ /* by id
*/ }
    // toString? equals? copy?
}
```

- A **Bulbasaur** has `id` (1=Bulbasaur, 2=Ivysaur, 3=Venusaur; start 1) and `level` (start 1).
  - `setLevel(int lv)` — updates level; [16,31] evolves (id→2); ≥32 → id→3.
  - `getName()` — returns the name from `id`.
  - **Then:** how would you write `toString()`, `equals()`, and `copy()`?

Diagnostic debrief — and where Class 2 goes

- **All three comfortable?** You are ready — the new material extends what you know.
- **Shaky on Q1–Q2?** Refresh 141/142 loops & arrays this week.
- **Q3 felt hard?** Normal — class design, `equals`, `toString`, `copy` are what CSC 240 is *for*.
- **Thursday (Class 2):** we solidify references, `.equals()`, static methods & ArrayList — then the engineering discipline (milestones, testing) and a preview of Chapter 9.

Before Thursday

- Install **IntelliJ IDEA** (Community Edition, free); write a small class with a constructor, a method, and a `main` that tests it.
- Take the online diagnostic if you have not: **wcu.ninja/csc240/diagnostic**.
- Read the **syllabus** (course site & D2L) — grading, the double-weighted HW3, academic-integrity + AI rules.
- Course site: **wcu.ninja/csc240**. No graded work yet — HW1 is assigned in Week 2.

Sources & further reading

- Microsoft 20–30% AI-generated code — Nadella, LlamaCon 2026 (Entrepreneur; Human Progress)
- Google 75% AI-generated code — Pichai, Google Cloud Next 2026 (Fast Company; DevOps.com)
- Veracode 2026 GenAI Code Security Report — 56% pass; XSS 15%, log injection 12% (veracode.com)
- Fortune-50 study: AI devs 3–4× commits, 10× security findings — CSA / industry reporting 2026
- METR randomized trial — experienced devs 19% slower, felt 20% faster (metr.org; The Register)
- OOP paradigm & encapsulation — adapted from the CSC 240 course materials (Gaddis Ch. 8–10)
- Diagnostic problems — CSC 141/142 Review set (GaussSum, Array, Bulbasaur)
- CSC 240 Fall 2026 Syllabus (course site & D2L)