

Class

10

CSC 240 Computer Science III

Mega Capstone + Midterm Review

Dr. Si Chen (schen@wcupa.edu)



Tue / Thu · 9:30 – 10:45 AM · Anderson Hall 315
Fall 2026 · Week 6 · Tuesday, September 29, 2026
wcu.ninja/csc240

Today

- **Part 1 (~30 min): finish the Mega Evolution capstone** we ran out of time for on Thursday. Charizard, Blastoise, [upgrade\(\)](#), and then the Pokémon World.
- **Part 2 (~40 min): midterm review, Chapters 9 and 10.** The midterm is **online and mostly programming** (a built-in code editor that runs tests), so most of today's review is writing code. There are also quick trace / "does it compile?" warm-ups.
- **Midterm: Thursday, Oct 8**, in class. **HW2 due Tuesday, Oct 6, 11:59 PM.** HW2 is the best practice for the Chapter 10 half of the exam.
- Code: the Class 9 zip ([05_mega_world_starter](#)) for Part 1 and the **Class 10 zip** for Part 2 ([wcu.ninja/csc240](#)).

PART 1

Mega Evolution, for Real

Thursday's capstone: Pokémon that really do have a Mega form. Finish the code, send your Pokémon to the class Pokémon World, and Mega Evolve it live on the big screen.

Recap: Venusaur is the worked example

```
public class Venusaur extends Pokemon
    implements MegaEvolvable {
    private boolean megaEvolved = false;

    public Venusaur(int hp, int atk, int def, int spd) {
        super(3, "Venusaur", "Grass", "Poison",
            hp, atk, def, spd, false);
    }
    @Override public void megaEvolve() {
        if (megaEvolved) return;
        attack += 18;        // protected in Pokemon
        defense += 40;
        megaEvolved = true;
    }

    @Override public void revertMega() {
        if (!megaEvolved) return;
        attack -= 18;        // undo EXACTLY
        defense -= 40;
        megaEvolved = false;
    }
    // + isMegaEvolved(), getMegaFormName()
}
```

- **extends Pokemon**: Venusaur **IS-A** Pokémon.
- **implements MegaEvolvable**: it **CAN** Mega Evolve, and the compiler now requires all 4 interface methods.
- **super(...)** must be the first line. The Pokémon part of the object gets built first.
- **attack** and **defense** are **protected**, so subclasses can change them directly.
- HP never changes in a real Mega Evolution. Only ATK, DEF, and SPD move.

Class 9 zip → 05_mega_world_starter/Venusaur.java

Your turn: Charizard, Blastoise, upgrade()

```
// Charizard.java - TODOs 1-6
public class Charizard extends Pokemon
/* TODO 1: implements MegaEvolvable */ {
    // TODO 2: private boolean field
    // TODO 3: megaEvolve()
    // TODO 4: revertMega()
    // TODO 5: isMegaEvolved(),
    //          getMegaFormName()
}

// PokedexApp.java - TODO 7
public static Pokemon upgrade(Pokemon p) {
    switch (p.getId()) {
        case 3: return new Venusaur(...);
        // case 6: Charizard    case 9: Blastoise
        default: return p;
    }
}
```

- Work out the boosts yourself (new stat minus CSV stat):
 - Mega Charizard X: ATK 130, DEF 111. CSV Charizard: ATK 84, DEF 78.
 - Mega Blastoise: ATK 103, DEF 120. CSV Blastoise: ATK 83, DEF 100.
- Blastoise has **no step-by-step hints**. It follows the same pattern as Venusaur.
- Without TODO 7, your classes are never constructed and no Mega button appears. Why? The CSV loader only creates plain [Pokemon](#) objects.

Stuck? Put Venusaur.java next to your file and compare them line by line.

Self-check, then send it to the World

```
Loaded 86 Pokemon.  
Can Mega Evolve:  
  Venusaur -> Mega Venusaur  
  Charizard -> Mega Charizard X  
  Blastoise -> Mega Blastoise  
  
// after megaEvolve():  
//   Mega Charizard X  ATK=130 DEF=111  
//   Mega Blastoise    ATK=103 DEF=120  
// after revertMega(): back to the CSV stats
```

- The console must list all three Pokémon before you go on.
- Open wcu.ninja/csc240/pokemonworld. On the Charizard card, click **Send to World** →, then enter your name, your ID, and the access code on the board.
- Click **Mega Evolve!**. Your Pokémon flashes on the projector and turns into its real Mega sprite. **Revert** turns it back.
- Bonus: Charizard Y, Gyarados, Mewtwo X/Y... (the stats table is in README.md).

Only Pokémon sent during this run of the app stay synced. If you restart the app, send them again.

What you just used = the midterm

Chapter 10 in your capstone

- `extends Pokemon`: inheritance, IS-A
- `super(6, "Charizard", ...)`: constructor chaining
- `attack += 46`: a protected field
- `implements MegaEvolvable`: an interface
- `@Override`: overriding, not overloading
- `ArrayList<Pokemon>` holding Charizards: parent reference, child object
- `p instanceof MegaEvolvable` + a cast: polymorphism

Chapter 9 in your capstone

- `line.split(",", -1)`: tokenizing a CSV row
- `f[4].trim()`: String methods
- `Integer.parseInt(...)`, `Boolean.parseBoolean(...)`: wrapper classes
- `String.format` in `toString()`
- `type2.isEmpty()`: blank tokens are still tokens

PART 2

Midterm Review

The midterm covers what we practiced in Classes 3–9. Today we do one example of each question type, then you know what to practice this week.

Midterm: what to expect

- **Thursday, October 8, in class (9:30–10:45), on your laptop, on a course web page:** wcu.ninja/csc240/midterm. It's 20% of your course grade.
- **Covers Chapters 9 and 10:** Classes 3–9, Quiz 1, HW1, and HW2. Thursday's new topic (lambdas) is **not** on it.
- **Part A: 10 multiple-choice questions (25 pts):** tracing, compile/run/crash, abstract vs. interface.
- **Part B: 2 programming problems (75 pts),** in a code editor built into the page (like an online judge):
 - a class hierarchy from a spec: abstract class, `super(...)`, an interface, `equals()` / `compareTo()` (like HW2)
 - a Chapter 9 text-processing class: `split` / `trim` / `parse` / `StringBuilder`
 - **Run** = compile + your own `Main` + a sample test. **Submit** = hidden tests, scored right away. Submit as often as you like; your **best** submission counts.
- **No AI tools**, no messaging, and no websites other than the exam page. It is individual work.

Try the practice exam this week

- **wcu.ninja/csc240/midterm/practice** is open now and **not graded**. Same page, same editor, same buttons as the real midterm.
- It has 5 MC questions plus two problems: **Vehicles** (today's Practice A) and **Text tools** (Chapter 9).
- Make sure it works on **your** laptop (Chrome, Edge, Firefox, or Safari): press Run, press Submit, and watch the tests turn green.
- Your code is saved in the browser as you type, and on the server every time you Run or Submit.
- Read the compiler errors on the page. A file that doesn't compile scores 0, so **compile early and often**.
- Charge your laptop and bring the charger. If you don't have a working laptop, **email me by Mon Oct 5**.

Chapter 9 at a glance

String & StringBuilder

- `String` is **immutable**. `s.toUpperCase()` returns a NEW string, so you must keep it: `s = s.toUpperCase();`
- `charAt`, `indexOf`, `substring(a, b)` (`b` is excluded), `trim`, `equals`, `equalsIgnoreCase`, `compareTo`
- Compare contents with `equals()`, never with `==`
- `StringBuilder` is **mutable**: `append`, `insert`, `reverse`, `deleteCharAt`, `setCharAt`, `toString()`

Wrappers, Character, tokens

- `Integer.parseInt("42")`, `Double.parseDouble`. Bad input throws `NumberFormatException`
- Autoboxing: `int` $\rightleftharpoons$ `Integer` automatically (e.g., `ArrayList<Integer>`)
- `Character.isDigit` / `isLetter` / `isWhitespace` / `toUpperCase`
- `s.split(",")` uses a regex, so a dot needs `split("\\. ")`. Also: `split(", ", -1)`, `StringTokenizer`, `Scanner`
- CSV row $\rightarrow$ tokens $\rightarrow$ parse $\rightarrow$ new object

Chapter 9: trace it

```
String s = "  pika,chu ,  rai chu  ";
String[] parts = s.trim().split(",");
System.out.println(parts.length);           // 1
System.out.println "[" + parts[2] + "];     // 2

StringBuilder sb = new StringBuilder(parts[0]);
sb.reverse().append('!');
System.out.println(sb);                     // 3

String t = parts[1];
t.toUpperCase();
System.out.println(t + t.length());         // 4

System.out.println(Character.isLetter(t.charAt(3))
    + " " + Character.toUpperCase(t.charAt(0))); // 5

int n = Integer.parseInt("4" + 2) + 1;
System.out.println(n);                     // 6

String x = "chu";
String y = new String("chu");
System.out.println((x == y) + " " + x.equals(y)
    + " " + x.equals(parts[1].trim()));     // 7
```

- Write down all 7 output lines **before** you look at the next slide.
- This is the midterm's "what does this print?" style.
- Careful with the spaces: `trim()` only removes them from the two **ends** of a string.

Class 10 zip → 01_ch9_trace/

Chapter 9: answers

```
3
[rai chu]
akip!
chu 4
false C
43
false true true
```

- **Line 1–2:** the pieces are "pika", "chu ", "rai chu". `trim()` runs once, on the whole string, **before** `split`. The pieces keep their own spaces.
- **Line 3:** `StringBuilder` methods change the object and return it, so the calls chain.
- **Line 4:** the #1 Chapter 9 trap. Calling `t.toUpperCase()`; without saving the result does nothing to `t`. "chu " has 4 chars. **Line 5:** `charAt(3)` is the space.
- **Line 6:** `+` with a `String` is concatenation.
- **Line 7:** `new String` creates a separate object, so `==` is false while `equals()` is true.

Chapter 10 at a glance

Inheritance

- `class B extends A`: B **IS-A** A and gets A's public/protected members
- Constructors are **not** inherited. `super(...)` must be the first line. If you leave it out, Java calls `super()` for you
- Construction runs **top-down**: `Object` → `A` → `B`
- `private` is class-only, `protected` is for subclasses (and the package), `public` is for everyone
- **Override** = same signature in the subclass (`@Override`). **Overload** = same name, different parameters
- `super.method()` calls the parent's version

Polymorphism & contracts

- `A ref = new B();`: the **reference type** decides what you may call. The **object type** decides which version runs (dynamic binding)
- Downcast `(B) ref` compiles. It throws `ClassCastException` if the object isn't really a B, so check with `instanceof`
- `abstract` class: can't use `new` on it; it can have abstract methods, fields, and constructors
- `interface`: a CAN-DO contract. A class can implement many
- From `Object`: `toString()`, `equals(Object)`. Plus `Comparable<T>` → `compareTo`

Chapter 10: trace it

```
class Animal {
    Animal() { System.out.println("Animal()"); }
    String name() { return "animal"; }
    String sound() { return "..."; }
    void speak() {
        System.out.println(name() + " says " + sound());
    }
}
class Dog extends Animal {
    Dog() { super(); System.out.println("Dog()"); }
    @Override String name() { return "dog"; }
    @Override String sound() { return "Woof"; }
}
class Puppy extends Dog {
    Puppy() { System.out.println("Puppy()"); }
    @Override String sound() {
        return super.sound() + " (tiny)";
    }
    void wag() { System.out.println("wag wag"); }
}
// main:
Animal a = new Puppy();
System.out.println("---");
a.speak();
```

- What does `main` print?
There are 5 lines.
- `Puppy()` doesn't write `super()`. Does `Dog()` still run?
- `speak()` exists only in `Animal`. So which `name()` and which `sound()` does it call?

Class 10 zip → 02_ch10_trace/

Chapter 10: answers

```
Animal()  
Dog()  
Puppy()  
---  
dog says Woof (tiny)
```

- **Constructors chain top-down.** Java inserts `super()` into `Puppy()`, `Dog()` calls `super()`, and so `Animal`'s constructor finishes first.
- **Dynamic binding:** `speak()` is `Animal`'s code, but `this` is a `Puppy` object. So `name()` finds `Dog`'s version (`Puppy` doesn't override it) and `sound()` finds `Puppy`'s.
- `super.sound()` inside `Puppy` runs **Dog's** `sound()` → "Woof", then " (tiny)" is added.
- The variable's type (`Animal`) doesn't matter for **which** method runs. It only matters for which methods you may **call**.

Compiles? Runs? Crashes?

```
Animal a = new Puppy();  
  
A: a.wag();  
B: ((Puppy) a).wag();  
C: Dog d = new Animal();  
D: Dog d2 = (Dog) a; d2.speak();  
E: Animal x = new Dog(); Puppy p = (Puppy) x;  
F: if (a instanceof Dog)  
    System.out.println("a IS-A Dog");
```

- For each line, decide: **compile error**, **runtime exception**, or **fine** (and what it prints).
- Rule: the compiler only looks at the **reference** type. The JVM looks at the **object**.
- Same classes as the previous slide.

Class 10 zip → 03_compiles_or_not/ (uncomment one line at a time)

Compiles? Runs? Crashes? Answers

- **A: compile error.** An Animal reference has no `wag()` method. The compiler doesn't care that the object is a Puppy.
- **B: fine** → `wag wag`. The cast makes the reference a Puppy, and the object really is one.
- **C: compile error.** Not every Animal is a Dog, so a parent object can't go into a child variable.
- **D: fine** → `dog says Woof (tiny)`. The object is still the same Puppy.
- **E: compiles, then crashes** with `ClassCastException`. The object is only a Dog, not a Puppy.
- **F: fine** → `a IS-A Dog`. A Puppy IS-A Dog, and `instanceof` checks the **object**.

Abstract class vs. interface

abstract class Shape

- Answers "what family is it?" (**IS-A**)
- You **extend** exactly one class
- Can have fields, constructors, and finished methods, **plus** abstract methods
- `new Shape()` is a compile error. `Shape s = new Circle(3);` is fine
- A subclass that doesn't implement every abstract method must itself be **abstract**
- e.g. `Employee` in HW2, `Shape`, `Vehicle`

interface Resizable

- Answers "what can it do?" (**CAN-DO**)
- You can **implement** any number of interfaces
- Its methods are **public abstract** by default: no body, no instance fields
- Unrelated classes can share one: `Employee` **and** `Invoice` are both `Payable` in HW2
- It works as a type: `Payable p = new Invoice(...); p instanceof MegaEvolvable`
- e.g. `Comparable<T>`, `MegaEvolvable`, `Payable`

Top Chapter 10 traps

```
// Trap 1: this OVERLOADS, it doesn't override!
public boolean equals(Circle other) { ... }
// correct:
@Override
public boolean equals(Object other) {
    if (this == other) return true;
    if (!(other instanceof Circle)) return false;
    Circle c = (Circle) other;
    return Double.compare(radius, c.radius) == 0;
}

// Trap 2: private is NOT visible in the subclass
class Car extends Vehicle {
    double x() { return plate.length(); } // error
}                                     // use getPlate() instead

// Trap 3: compareTo sign
return Double.compare(this.area(), other.area());
//      negative -> this comes first
```

- `equals` must take `Object`. Always write `@Override` so the compiler catches a wrong parameter type.
- If the parent has **no** no-arg constructor, the child **must** call `super(args)` explicitly.
- Abstract method: `public abstract double area();` has a semicolon and no body.
- An interface method in a class must be `public` (you can't make its access weaker).
- To reverse an order, swap the arguments:
`Double.compare(other.area(), this.area())`.

Practice A: Vehicle, Car, Truck, Inspectable

```
Vehicle[] fleet = {  
    new Car("PA-1234"),  
    new Truck("PA-7777", 3),  
    new Truck("PA-9000", 6),  
};  
// Expected output:  
// Car PA-1234 toll for 100 miles: $10.00  
// Truck PA-7777 toll for 100 miles: $75.00  
// Truck PA-9000 toll for 100 miles: $150.00  
// Total: $235.00  
// Failed inspection: Truck PA-9000  
// Same vehicle? true
```

- `interface Inspectable`
`{ boolean`
`passesInspection(); }`
- `abstract class Vehicle`: private
`plate`, a constructor, `getPlate()`,
`abstract double tollFor(int`
`miles)`, `toString()` → "Car
PA-1234", and `equals` by plate
- Car: \$0.10 per mile
- Truck extends Vehicle
implements `Inspectable`: \$0.25
per axle per mile; passes inspection
only with ≤ 5 axles
- Bonus: `Comparable<Vehicle>` by
`tollFor(100)`

Also Problem 1 of the practice exam:
wcu.ninja/csc240/midterm/practice (timer on: about 25 min)

Practice B: load the fleet from a CSV (Chapter 9)

```

type,plate,axles
Car, pa-1234 ,
Truck,PA-7777,3
truck,PA-9000,6
Car,PA-1234,

Truck,PA-5150,x
Car,NJ-0042,

// Expected output of FleetLoader:
// Skipped duplicate: PA-1234
// Skipped bad row: Truck,PA-5150,x
// Loaded 4 vehicles:
//   Car PA-1234
//   Truck PA-7777
//   Truck PA-9000
//   Car NJ-0042
// Total toll for 100 miles: $245.00

```

- Finish `parse(line)` and the read loop in `FleetLoader.java`.
- `split(",", -1)`, then `trim()` and `toUpperCase()` the plate. Accept `Car` or `Truck` in any case with `equalsIgnoreCase`.
- Skip blank lines. Catch `NumberFormatException` from `Integer.parseInt("x")`.
- Skip duplicates with `fleet.contains(v)`. It calls **your** `equals()`, so this is where a wrong `equals` shows up.
- This is the mix the midterm uses: Chapter 10 classes + Chapter 9 text processing, in one program.

Class 10 zip → 04_write_it_starter/FleetLoader.java + fleet.csv

How to study this week

- **Finish HW2** (due Tue Oct 6) **by yourself, without AI**. It is the closest thing to the midterm's programming part.
- Go over the **Quiz 1 answers & explanations** (wcu.ninja/csc240) and your **HW1** feedback for Chapter 9.
- Re-run the class code for **Classes 3–9**. Predict each output before you run it.
- Take the **practice exam** (wcu.ninja/csc240/midterm/practice) **with a timer**. Do Practice B in IntelliJ. Then rebuild HW2's Employee hierarchy from scratch without looking.
- Get fast at the basics: writing a constructor with `super(...)`, `@Override`, `equals(Object)`, `compareTo`, split/parse. Speed matters on a timed programming exam.
- Also review the traps (`==` vs `equals`, immutable Strings, overloading vs overriding, casts) for the short questions.
- Questions: office hours Tue/Thu 12:30–2:00 (by appointment), Wed 1:00–3:00, or email with the exact code/error.

Sources

- Gaddis, **Starting Out with Java**, 4th ed., Ch. 9 (text processing, wrapper classes) and Ch. 10 (inheritance, polymorphism, abstract classes, interfaces).
- Capstone: CSC 240 Class 9 Mega Evolution starter. Mega stats from the official Pokémon X/Y data (via PokéAPI); sprites from the PokéAPI sprite CDN.
- Review problems (Animal/Dog/Puppy, Vehicle/Car/Truck, fleet.csv) written for this class. Outputs were verified with OpenJDK 17.
- Java SE 17 API: String, StringBuilder, Character, Integer, Object, Comparable.