Class

6

CSC 240 Computer Science III

Chapter 10: Inheritance Fundamentals

Dr. Si Chen (schen@wcupa.edu)

Tue / Thu · 9:30 – 10:45 AM · Anderson Hall 315
Fall 2026 · Week 4 · Tuesday, September 15, 2026
HW1 due tonight, 11:59 PM · wcu.ninja/csc240

PART Review

Let’s go over the Pokédex

Class 5’s CSV → objects → GUI → Pokémon World, together

The solution: Pokemon.fromCsv(line)

```
public static Pokemon fromCsv(String line) {
    // split(",", -1) keeps a trailing EMPTY field
    // (a single-type Pokemon's type2 column)
instead
    // of silently dropping it.
    String[] f = line.split(",", -1);
    int id = Integer.parseInt(f[0].trim());
    String name = f[1].trim();
    String type1 = f[2].trim();
    String type2 = f[3].trim();
    int hp = Integer.parseInt(f[4].trim());
    int attack = Integer.parseInt(f[5].trim());
    int defense = Integer.parseInt(f[6].trim());
    int speed = Integer.parseInt(f[7].trim());
    boolean legendary =
        Boolean.parseBoolean(f[8].trim());
    return new Pokemon(id, name, type1, type2,
        hp, attack, defense, speed, legendary);
}
```

Same shape as `Person.fromCsv` (Class 4) and `GasPrices.java` (HW1 Q3): split the line, `parseInt`/`parseBoolean` the fields, return a new object.

The `split(",", -1)` gotcha bit a few people - without `-1`, a trailing empty `type2` silently vanishes and every field after it shifts over by one.

Then `PokedexGUI.launch(...)` / `.showAll(...)` (given, not written by you) turned the list into cards with stat bars.

The Pokémon World, by the numbers

86

Pokémon in pokemon_data.csv

the starter dataset, Gen 1 only

9

students sent a Pokémon

out of 31 - if you did not get to it, the code still works

151+

IDs actually accepted

the server checks anything past #151 against PokeAPI - try it

wcu.ninja/csc240/pokemonworld is still live. Click any Pokémon there to see its stats and level - `Pokemon.setLevel(int)` was the optional bonus method; today's new class is what makes a "leveled-up, specialized" Pokémon possible as its own real class.

Quick poll: what tripped people up?

Forgetting to **skip the header line** before the read loop.

A stray **whitespace** in a CSV field breaking `Integer.parseInt` with a `NumberFormatException`.

On newer macOS + old JDK 17: a noisy but harmless `accessibilityHitTest` crash trace on every click (fixed in the starter with one `System.setProperty(...)` line - if you retyped `main()` by hand you may have dropped it).

Forgetting the **`-1`** in `split(",", -1)` and getting a shifted or missing field.

PART 1

From one class... to a family of classes

The Pokémon class works great. What about a Pokémon that needs EXTRA behavior?

The problem: related classes, duplicated code

Imagine writing `LegendaryPokemon`, `EvolvedPokemon`, `StarterPokemon` - each needs id, name, hp, attack, defense, speed... and you would copy-paste the SAME fields and getters into every one.

Copy-pasted code is a maintenance trap: fix a bug in one copy, forget the other three.

All of these classes really are "a kind of" `Pokemon` - they share a common core and each adds a bit more. That relationship is exactly what **inheritance** is for.

This is the same DRY principle behind writing `Pokemon.fromCsv` ONCE instead of copy-pasting parsing code into every program that reads the CSV.

Superclass and subclass: extends

```
public class Pokemon {
    private int id;
    private String name;
    protected int hp, attack, defense, speed;
    // constructor, getters, toString() ...
}

public class Eevee extends Pokemon {
    private String evolutionStone;

    public Eevee(int id, String name,
                 int hp, int attack,
                 int defense, int speed) {
        super(id, name, "Normal", "",
              hp, attack, defense, speed, false);
        this.evolutionStone = "";
    }
}
```

`Pokemon` is the **superclass** (a.k.a. base/parent class); `Eevee` is the **subclass** (a.k.a. derived/child class).

`extends Pokemon` means: Eevee automatically has every **public** method Pokemon has - `getHp()`, `getName()`, `toString()` - without retyping a single line.

`super(...)` calls Pokemon's constructor **first**, to build the "Pokemon half" of the object, before Eevee's own constructor code runs.

This IS-A test is the giveaway: "an Eevee IS A Pokemon" reads naturally in English -> inheritance is the right tool.

Full listing: code/class06_inheritance/Pokemon.java and Eevee.java.

What exactly does a subclass inherit?

Public members: yes, directly usable, e.g. `eevee.getHp()`.

Protected members: yes, directly usable from INSIDE the subclass's own code (like `hp` inside a method in `Eevee`) - see the next slide.

Private members: they still exist in memory (every Eevee object DOES have Pokemon's private `id` and `name`), but the subclass cannot refer to them **by name** - only through inherited public methods like `getId()`.

Constructors are **not** inherited - every subclass writes its own constructor (which usually calls `super(...)`).

PART 2

The protected access modifier

Family-only access: not as open as public, not as locked as private

Access modifiers, side by side

private

Only this class

Not even subclasses

Use for: fields you always want accessed through getters/setters

protected

This class + subclasses

Not unrelated classes

Use for: fields a subclass genuinely needs to read/write directly

public

Anyone, anywhere

No restriction

Use for: the class's real interface - methods, not usually fields

Why hp/attack/defense/speed became protected

```
// BEFORE (Class 5 Pokedex): all private
private int hp, attack, defense, speed;

// AFTER (today): protected
protected int hp, attack, defense, speed;

// Now Eevee can update stats DIRECTLY,
// no getter/setter round trip needed:
public void useStone(String stone) {
    this.evolutionStone = stone;
    hp += 10; attack += 5;
    defense += 5; speed += 5;
}
```

`id`, `name`, `type1`, `type2`, `legendary` **stayed private** - no subclass in this deck needs to touch them directly, so there is no reason to loosen their access.

Only the fields a subclass genuinely needs (the battle stats, for an evolution-style stat boost) became `protected`.

Rule of thumb: default to `private`; loosen to `protected` only when a subclass has a real, concrete need to reach in directly.

In-class: extend the family

IN-CLASS 10 min

Starter files: `code/class06_inheritance/Pokemon.java` and `Eevee.java` (both already given).

Write a new subclass `Vaporeon extends Pokemon` (or reuse `Eevee` and adapt it) that adds one new field and one new method, the same way `Eevee.useStone(...)` does.

In `InheritanceDemo.java`, create one of your new Pokémon and call both an inherited method (`getHp()`) and your new method.

Checklist: `extends Pokemon` · `super(...)` as the first line of the constructor · a new field the superclass does not have · a new method that uses a `protected` field directly.

Stretch: also look at `Vehicle.java`/`ElectricCar.java` in the same folder - same rules, a non-Pokémon example.

Skeleton:

```
public class Vaporeon
    extends Pokemon {
  private boolean
hasAquaRing;

  public Vaporeon(int id,
      String name, int hp, ...) {
    super(id, name, "Water", "",
        hp, ...);
  }

  public void useAquaRing() {
    hasAquaRing = true;
    hp += 15; // protected field
  }
}
```

Before next class (Thu, Sep 17)

HW1 is due tonight, 11:59 PM on D2L - this is the priority for tonight.

Read **Gaddis Chapter 10 §10.1-10.2** (inheritance basics, the `protected` modifier) before Thursday.

Thursday: multi-level hierarchies (`Jolteon extends Eevee extends Pokemon`), is-a vs has-a, and more practice with `protected`.

Course site: **wcu.ninja/csc240** - Class 6 code zip is posted.

Sources

Gaddis, Starting Out with Java, 4th ed., Ch. 10 §10.1-10.2 (inheritance, superclass/subclass, protected).

Pokemon.java, Eevee.java: built on the CSC 240 Fall 2026 Pokédex capstone (Class 5); Vehicle.java/ElectricCar.java: standard textbook-style inheritance example.

Eevee/Jolteon as a class-hierarchy example: continues the bridge slide from Class 5 (Eevee evolution chart, Dr. Si Chen CSC 240 Spring 2019 Class 3 deck; species data from PokeAPI).

Java SE 17 API: Object, access control (Java Language Specification §6.6).