Class

5

# CSC 240 Computer Science III

## Integrated Text Processing

**Dr. Si Chen (schen@wcupa.edu)**

Tue / Thu · 9:30 – 10:45 AM · Anderson Hall 315
Fall 2026 · Week 3 · Thursday, September 10, 2026
Quiz 1 today (first 15 min) · HW1 due Tue Sep 15, 11:59 PM ·
wcu.ninja/csc240

# Quiz 1 — 15 minutes, closed notes

Chapter 9: wrapper classes & `parseInt`/`parseDouble`, autoboxing, `Character` tests, String search/extract methods, `StringBuilder` vs `String`, `split` / `StringTokenizer`.

Format: short answers, "what does this print", one small method to write.

No notes, no AI, no phones. Show your reasoning for partial credit.

Done early? Re-read HW1 Q3 and list the steps your `GasPrices` needs — it is due **Tuesday 11:59 PM**.

# Recap from Tuesday

`split(regex)` → `String[]`; `StringTokenizer` → one token at a time; `Scanner` → typed tokens from any source.

The delimiter is a **regex**: `"\\s+"` for whitespace, `"\\."` for a literal dot.

Tokens are **text** until `Integer.parseInt` / `Double.parseDouble` — `trim()` first, validate or catch `NumberFormatException`.

Parse at the boundary, then work with **objects** (`Person.fromCsv`).

Today: **the toolkit, assembled** — five problems, five patterns.

# The Chapter 9 toolkit on one slide

## Test a char

`Character.isDigit(c)`

`isLetter` `isUpperCase`

`isLetterOrDigit` `isWhitespace`

`toUpperCase(c)` `toLowerCase(c)`

## Search & extract

`indexOf(x)` `lastIndexOf(x)` (−1 if absent)

`startsWith` `endsWith` `contains`

`substring(b, e)` — `e` exclusive

`charAt(i)` `length()`

## Edit text

`StringBuilder sb`

`append` `insert` `setCharAt`

`deleteCharAt` `replace` `reverse`

`sb.toString()` at the end

## Tokenize

`s.split(regex)` → `String[]`

`new StringTokenizer(s, delims)`

`new Scanner(s).useDelimiter(...)`

Watch for empty tokens & headers

## Convert

`Integer.parseInt(t.trim())`

`Double.parseDouble(t)`

`String.valueOf(n)`

`NumberFormatException` on bad text

## Compare

`a.equals(b)` — **never** `==`

`equalsIgnoreCase`

`compareTo` → <0, 0, >0

`toLowerCase()` before comparing

PART 1

# Pattern: scan with a flag

Walk the string character by character; a boolean remembers the state

# Sentence Capitalizer

```
public static String capitalize(String s) {
    StringBuilder sb = new StringBuilder(s);
    boolean startOfSentence = true;        // the FLAG
    for (int i = 0; i < sb.length(); i++) {
        char c = sb.charAt(i);
        if (startOfSentence
              && Character.isLetter(c)) {
            sb.setCharAt(i,
                Character.toUpperCase(c));
            startOfSentence = false;
        } else if (c == '.' || c == '!'
                   || c == '?') {
            startOfSentence = true;
        }
    }
    return sb.toString();
}
// "the fox. it jumped!" -> "The fox. It jumped!"
```

The **flag** `startOfSentence` carries information from one character to the next.

`StringBuilder.setCharAt` edits **in place** — impossible on an immutable `String`.

Skips spaces after a period naturally: the flag stays up until a **letter** arrives.

Pattern reuse: "first letter of each word" is the same loop with a **space** resetting the flag.

*SentenceCapitalizer.java (Gaddis Ch. 9 problem, my version).*

PART 2

# Pattern: several rules, one pass

Collect facts in one loop, decide at the end

# Password Verifier

```
// Rules: >= 8 chars, one upper, one lower, one digit
public static boolean isValid(String pw) {
    if (pw.length() < 8) return false;   // cheap check
first
    boolean hasUpper = false, hasLower = false,
            hasDigit = false;
    for (int i = 0; i < pw.length(); i++) {
        char c = pw.charAt(i);
        if (Character.isUpperCase(c)) hasUpper = true;
        else if (Character.isLowerCase(c)) hasLower = true;
        else if (Character.isDigit(c)) hasDigit = true;
    }
    return hasUpper && hasLower && hasDigit;
}
```

Three flags, **one loop** — not three loops. Each character updates whichever fact it proves.

The **early return** on length avoids scanning a hopeless string.

Returning a `boolean` keeps the method **testable**: `isValid("Secret123")` → `true`; `isValid("secret")` → `false`.

Extending the rule (a special character?) = one more flag, one more `else if`.

*PasswordVerifier.java — Gaddis Ch. 9 programming challenge, HW-style.*

PART 3

# Pattern: strip → validate → rebuild

Normalize messy input into one canonical form before formatting it

# Phone number formatter

```
String[] inputs = { "610-436-6998",
    "(610) 436 6998", "6104366998", "610.436.69" };
for (String raw : inputs) {
    String digits = keepDigits(raw);     // STRIP
    if (digits.length() == 10)           // VALIDATE
        System.out.printf("(%s) %s-%s%n", // REBUILD
            digits.substring(0, 3),
            digits.substring(3, 6),
            digits.substring(6));
    else System.out.println(raw + " -> invalid");
}
static String keepDigits(String s) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < s.length(); i++)
        if (Character.isDigit(s.charAt(i)))
            sb.append(s.charAt(i));
    return sb.toString();
}
```

*PhoneFormatter.java.*

Three different spellings of the department phone number → **one** canonical digit string → **one** output format.

`keepDigits` is a reusable helper: **filter** characters into a `StringBuilder`.

`substring(b, e)`: `e` is **exclusive**; `substring(6)` runs to the end.

HW1 Q2 (`Word2Number`) is this pattern with a `letterToDigit` step in the middle.

PART 4

# Pattern: tokenize, then measure

Words, counts, longest, frequency — statistics over tokens

# Word statistics

```
String text = "It was the best of times, "
            + "it was the worst of times;";
String[] words = text.toLowerCase().split("[^a-z']+");

int total = 0, longestLen = 0, countIt = 0;
String longest = "";
for (String w : words) {
    if (w.isEmpty()) continue;             // split may
leave ""
    total++;
    if (w.length() > longestLen)
        { longestLen = w.length(); longest = w; }
    if (w.equals("it")) countIt++;         // .equals —
NEVER ==
}
// words: 12  longest: "times"  "it" x 2

// 2019 review: wordCount("CSC 240 Si Chen") -> 4
static int wordCount(String s)
    { return s.trim().split("\\s+").length; }
```

Delimiter `"[^a-z']+"` = one or more characters that are **not** letters or apostrophes — punctuation vanishes.

`toLowerCase()` **before** splitting so `"It"` and `"it"` count together.

`w.equals("it")`: `==` compares **references** — two equal strings from `split` are usually different objects, so `==` silently fails.

My 2019 `WordCounter`: the same idea in one line.

*WordStats.java — includes average word length.*

PART 5

# Pattern: a real file, messy lines

The Test Score Reader: skip, validate, average — without crashing

# TestScoreReader.java + scores.csv

```
// scores.csv: Lovelace,Ada,92,88,95
//             Hamilton,Margaret,88,x,91
Scanner file = new Scanner(new File("scores.csv"));
while (file.hasNextLine()) {
    String line = file.nextLine().trim();
    if (line.isEmpty()) continue;        // blank
    String[] f = line.split(",");
    if (f.length < 3) continue;          // too short
    String name = f[1].trim() + " " + f[0].trim();
    double sum = 0; int n = 0;
    for (int i = 2; i < f.length; i++) {
        String tok = f[i].trim();
        if (isNumeric(tok))
            { sum += Double.parseDouble(tok); n++; }
        else System.out.println("bad score " + tok);
    }
    System.out.printf("%-20s avg %.1f%n",
                      name, n == 0 ? 0 : sum / n);
}
file.close();
```

My 2019 version: "Dr. Chen exports grades from Excel/D2L as CSV; sum each student's scores." Same shape.

Real files have **blank lines, short lines, bad tokens** (`x`). Each gets a rule — the program **never crashes**.

`isNumeric` (digits and one dot) from Tuesday guards every `parseDouble`.

`printf("%-20s")` left-aligns names in a column; `%.1f` one decimal.

*Run from the folder that contains scores.csv. Output shows "ignoring bad score x" for Hamilton.*

# Review the AI-generated code: three flaws

## An AI wrote this

```
// Task: count how many words in `text` equal
"java"
String[] w = text.split(".");
int count = 0;
for (int i = 0; i <= w.length; i++) {
    if (w[i] == "java") count++;
}
System.out.println(count + " matches");

// It compiles. It is wrong in three ways.
```

## Your review

**Find them (3 min, pairs):**

1. `split(".")` — regex dot = any char → **empty array**.
2. `i <= w.length` — **off by one** → `ArrayIndexOutOfBounds`.
3. `w[i] == "java"` — **reference** comparison → almost always `false`.

Fix: `split("\\s+")`, `i < w.length`, `w[i].equalsIgnoreCase("java")`.

None of these is a compile error. This is why **you** review the code and **test with a known answer** — the pilot rule from Class 2.

# In-class: search Pokémon by name prefix

IN-CLASS 12 min

```
Mac
66:"Machop"
67:"Machoke"
68:"Machamp"
```

`pokemonName.csv` (Class 5 zip): header `"name",` then one quoted name per line; line 2 = Pokémon **#1**.

**Write PokemonSearch:** read the names into an `ArrayList<String>` (strip quotes + trailing comma); ask for a prefix; print `id:"Name"` for every match, case-insensitive.

Expected for `Mac`: `66:"Machop" 67:"Machoke" 68:"Machamp"` (my 2019 review question).

**Checklist:** skip the header · `split(",")[0]` · `replace("\"", "")` · `startsWith` · id = index + 1.

**Stretch:** a *contains* search — which String method?

Skeleton:

```
f.nextLine();   // header
while (f.hasNextLine()) {
  String s = f.nextLine().trim();
  names.add(s.split(",")[0]
      .replace("\"", ""));
}
for (i = 0; i < names.size(); i++)
  if
(names.get(i).toLowerCase()
      .startsWith(prefix))
    print((i+1) + ":\"" + ...);
```

# Capstone: build a Pokédex



`pokemon_data.csv` (starter zip, 86 Pokémon): id, name, two types, four stats, legendary flag — one line per Pokémon.

**Write `Pokemon.fromCsv(line)`:** split on `","`, `Integer.parseInt` the stats, `Boolean.parseBoolean` the last field. Same shape as `Person.fromCsv` from Class 4 — text in, object out.

**Write the read loop** in `PokedexApp`: open the file, skip the header, call `fromCsv` per line, collect into an `ArrayList<Pokemon>`.

Then call the **given** `PokedexGUI.launch(...)` / `.showAll(...)` — a window pops up: one card per Pokémon, type-colored badge, HP/ATK/DEF/SPD bars. You never write GUI code.

**Bonus, not required:** each card has a "Send to World →" button — pick a Pokémon, type your name + student ID, and it walks onto **wcu.ninja/csc240/pokemonworld**, a shared field with everyone else's picks. Click one there to see its stats.

Skeleton (fromCsv):

```
String[] f = line.split(",");
int id = Integer.parseInt(f[0]);
String name = f[1];
...
boolean legendary =
    Boolean.parseBoolean(f[8]);
return new Pokemon(id, name,
...);
```

Then:

```
PokedexGUI.launch("My
Pokedex");
PokedexGUI.showAll(pokedex
);
```

# Why the fun data matters: Eevee has nine forms


*Eevee evolution chart from my 2019 Class 3 deck — our running example for Chapter 10.*

Next week: **Chapter 10 — Inheritance**. `Eevee` evolves into `Vaporeon`, `Jolteon`, `Flareon`, … each **is-a** Pokémon with extra behavior.

You parsed Pokémon **names** today; next week you model Pokémon **types** as a class hierarchy — `Pokemon` → `Eevee` → `Jolteon`.

Reading: Gaddis Ch. 10 §10.1–10.3 (superclass, subclass, `extends`, `super`).

The diagnostic Q3 (`Bulbasaur` with `setLevel` evolution) was the warm-up for exactly this.

# Before next class (Tue, Sep 15)

**HW1 due Tuesday, Sep 15, 11:59 PM** on D2L — starter zip on the course site; do not edit template code; style counts (−5).

Finish `PokemonSearch` and the **Pokédex capstone** if you did not; reference solutions are in the Class 5 zip / pokedex bundle.

Haven't sent a Pokémon yet? **wcu.ninja/csc240/pokemonworld** stays interesting to check on even after class.

Read **Gaddis Chapter 10 §10.1–10.3** (inheritance basics). Bring one real-world *is-a* example to class.

Quiz 1 grades on D2L by Tuesday. **HW2** (inheritance) is assigned next week.

Course site: **wcu.ninja/csc240** — Class 5 code zip (5 patterns + Pokémon data) and the Pokédex starter zip are posted.

# Sources


Gaddis, Starting Out with Java, 4th ed., Ch. 9 — Sentence Capitalizer, Password Verifier, TestScoreReader problems (adapted).

Word Counter ("CSC 240 Si Chen" → 4), Process Score CSV problem, Search Pokémon ID exercise and sample run: Dr. Si Chen, CSC 240 Spring 2019 Class 6–7 review decks.

Eevee evolution chart and pokemonName.csv: Dr. Si Chen, CSC 240 Spring 2019 Class 3 deck / course files (data from PokeAPI).

Pokédex capstone (pokemon_data.csv, PokedexGUI, Pokémon World): built for CSC 240 Fall 2026. Sprites: PokeAPI public sprite CDN (pokeapi.co), used for this single-class, non-commercial demo.

Java SE 17 API: String, StringBuilder, Character, Scanner, java.util.regex, java.net.http.

Oracle Java Tutorials, "Comparing Strings" (equals vs ==).