Unit 4: Data Collections
Using Text Files

Aligned to:

AP Computer Science A Course and Exam Description

Effective Fall 2025

Topic 4.6

This work is licensed under the
Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

https://longbaonguyen.github.io

Why Files?

- So far every data set you have used was typed into the program itself.
- That has a hard limit: to change the data, you must edit and recompile the code. And you can only ever include as much data as you are willing to type.
- A file separates the two. The program stays the same; the data changes.
- This is how most programs get their data, and it is new to the AP course as of Fall 2025.

Two Classes Only

- Reading a file on this exam uses exactly two classes:

```
File      names a file on disk
Scanner   reads from it
```

- You already know Scanner. It is the same class you used for keyboard input; you just hand it a File instead of System.in.
- Everything you need is on the exam reference sheet. If it is not on that sheet, you are not expected to use it.

Opening a File

- Two imports, then one line:

```
import java.io.File;
import java.util.Scanner;
File f = new File("data.txt");
Scanner input = new Scanner(f);
```

- Read that inside out: make a File that names data.txt, then make a Scanner that reads from it.
- When you are finished, release it:

```
input.close();
```

throws IOException

- A file might not be there. Java forces you to acknowledge that before it will compile.
- The way you acknowledge it in this course is on the method header:

```
public static void readData() throws IOException
```

- That says: this can fail, and I am not handling it here. If the file name turns out to be invalid, the program terminates. File and IOException both live in java.io, so both need importing.
- Java also offers try/catch for this, but try/catch is outside the scope of the AP course. Use throws.

The Reading Loop

- You do not know how many lines a file has, so instead of a counted loop you ask whether anything is left:

```
while (input.hasNext()) {
    String line = input.nextLine();
    System.out.println(line);
}
input.close();
```

- hasNext() returns false once there is nothing more to read, which is what ends the loop.

Scanner Methods

- From the exam reference sheet, in full:

```
boolean hasNext()     is there more to read?
String  nextLine()    read the rest of the line
String  next()        read one token
int     nextInt()     read one int
double  nextDouble()  read one double
boolean nextBoolean()
void    close()
```

- That is the complete list for this course.

Lines or Tokens

- There are two ways to read, and they do not mix comfortably.
- BY LINE: nextLine() takes everything up to the end of the line, newline included.
- BY TOKEN: next(), nextInt(), nextDouble() take one whitespace-separated piece and stop, leaving the newline behind.
- Mixing them leaves that stray newline in the way and hands you an empty string where you expected data. The CED excludes this outright: "Writing or analyzing code that uses both nextLine and other Scanner methods on the same input source is outside the scope."
- Decide which one a file wants, and use only that one.

Splitting a Line

- A line read with nextLine() arrives as one String. To get at the fields, split it on whatever separates them:

```
String line = "Alicia,92,88,100";
String[] parts = line.split(",");
// parts[0] is "Alicia"
// parts[1] is "92"   <-- a String, not an int
```

- Numbers read from a file are text until you convert them:

```
int score = Integer.parseInt(parts[1]);
```

Example: scores.txt

- A file with one student per line, giving a name and then three test scores:

```
Alicia,92,88,100
Ben,75,80,68
Chen,88,95,91
Dara,60,72,85
```

- Goal: print each student's name and average.
- Before writing code, say the plan: read a line, split it on commas, convert parts 1 onward to ints, total them, divide by how many there were.

The Code

```
public static void report(String file)
        throws IOException {
  Scanner in = new Scanner(new File(file));
  while (in.hasNext()) {
    String[] parts = in.nextLine().split(",");
    int sum = 0;
    for (int i = 1; i < parts.length; i++)
      sum += Integer.parseInt(parts[i]);
    System.out.println(parts[0] + ": " +
        (double) sum / (parts.length - 1));
  }
  in.close();
}
```

Storing What You Read

- Printing as you go works, but usually you want to keep the data and use it later.
- You do not know how many lines there are, so an array is awkward. Read into an ArrayList:

```
ArrayList<String> names =
        new ArrayList<String>();
while (in.hasNext()) {
    String[] parts = in.nextLine().split(",");
    names.add(parts[0]);
}
```

- Every algorithm you already know (search, count, max, sort) then works on it.

What Is NOT on the Exam

- Java can do far more with files than this. None of it is assessed. These are the CED's own exclusions:
- WRITING to files. There is no FileWriter and no PrintWriter here, only reading.
- BufferedReader and FileReader.
- try/catch. Use throws instead.
- hasNextLine(), hasNextInt() and hasNextDouble(). Only hasNext() is in scope.
- The java.nio.file classes. Note also that "Accepting input from the keyboard is outside the scope", so here Scanner is used for files.
- If a method is not on the reference sheet, it is not expected of you.

Lab

- Create a file called scores.txt with at least six students, each on their own line, in the form name,score,score,score.
- Then write these methods, each declaring throws IOException:
- 1) printAverages(String filename): prints the name and average for each student.
- 2) topStudent(String filename): returns the name of the student with the highest average.
- 3) countAbove(String filename, double cutoff): returns how many students averaged above the cutoff.
- Test each one with an empty file and with a single-student file before you call it finished.

References

1) Building Java Programs: A Back to Basics Approach by Stuart Reges and Marty Stepp

2) Runestone CSAwesome Curriculum: https://runestone.academy/runestone/books/published/csawesome/index.html

For more tutorials/lecture notes in Java, Python, game programming, artificial intelligence with neural networks:

https://longbaonguyen.github.io