COMP249 Tutorial 7: More on File I/O and Recursion

Ella Noyes

July 31, 2024

1 Approaches to File I/O

There are many approaches to reading from and writing to a file. The approach that you choose depends on the application.

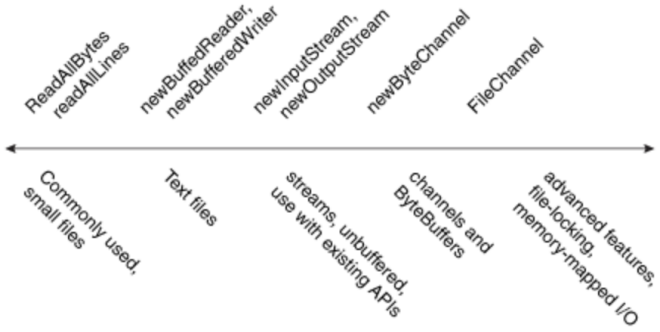

Figure 1: Figure is from: https://docs.oracle.com/javase/tutorial/essential/io/file.html

We've been experimenting with

- Text I/O: Using Scanner, BufferedReader, and BufferedWriter
- Stream I/O (a.k.a. Binary I/O, Byte Stream I/O, ...): Using InputStream and OutputStream
- Object Serialization: Using ObjectInputStream and ObjectOutputStream

Text I/O is built on stream I/O, but it performs encoding/decoding operations. I found a useful article comparing text I/O and stream I/O

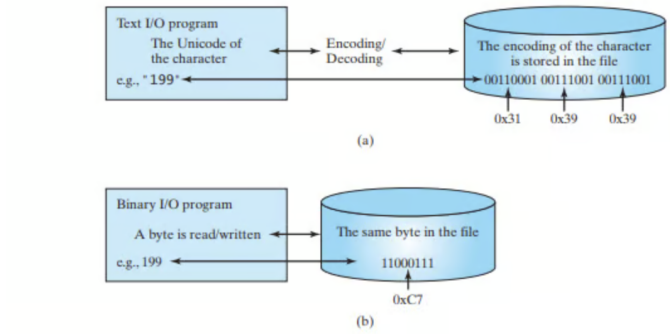

Figure 2: Figure is from the article above

When you write the number 199 to a file using text I/O (using BufferedWriter, for example), it will be stored as those three characters '1', '9', '9'. Whereas the stream I/O approach allows you to store 199 directly as its binary value 11000111.
A binary file consists of raw binary data, and is not marked by an end of file (oef) character like a text file. Read more on binary file I/O here.

I will show you how this looks in my IDE. I'm using an IntelliJ plugin called BinEd to show you what those different files look like in binary.

Anyways, here's what ChatGPT says about when to use stream I/O versus text I/O:

- **Stream I/O (Binary I/O)** is best for handling binary data, non-textual content, and custom formats. It provides efficient low-level access to data and is suitable for cases where you need to handle raw byte data or serialized objects.
- **Text I/O is** ideal for handling character-based data, where text encoding and decoding are important. It simplifies reading and writing textual data, making it suitable for applications that process text files or need to handle different character encodings.

2 Recursion

Recursion occurs when a method calls itself. It can be used to break a problem down into smaller parts. Recursion is referred to as a "Divide and Conquer" approach to problem-solving. A recursive method has two properties:

- **A base case:** The case that breaks out of the recursive method calls (i.e., a scenario that contains a return statement).
- **A recursive step:** A rule that leads successive method calls towards the base case.

2.1 Recursion Example: Find n!

Reminder: n's factorial is n! = n * (n-1) * (n-2) * ... * 1, where n is a non-negative integer. We can break up that product using recursion. Let's see how that would look in code.

2.2 Recursion Example: Finding the n-th Term of the Fibonacci series

A classic example of a recursion problem requires finding the n-th term of the Fibonacci series. In the fibonacci series, every number is the sum of the two previous numbers in the sequence. It goes: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,... The 8th term of this sequence is 21, which is 8 + 13. We can find that term by building up a sum of the previous terms (0 + 1 + 1 + 2 + ... + 13), and we can build up that sum using recursion!
To calculate the n-th term of the fibonacci series using recursion:

- Define a method fibonacci(num) that returns fibonacci(num - 1) + fibonacci(num - 2)
- The last two terms of the fibonacci serie, 0 and 1, should be used as the base cases.

A bulky (but maybe easier to start with!) version of that method would be

```
public int fibonacci(int num) {
    // Base case 1
    if(num == 0) {
        return 0;
    }

    // Base case 2
    if(num == 1) {
        return 1;
    }

    return fibonacci(num - 1) + fibonacci(num - 2);
}
```

But you could clean that up by combining those base cases

```
public int fibonacci(int num) {
    if(num < 2)
        return num;

    return fibonacci(num - 1) + fibonacci(num - 2);
}
```

2.3 Recursion Example: Check Whether a String is a Palindrome

A palindrome is a word or phrase that reads the same way backwards and forwards (e.g., kayak, eye, level, "was it a cat i saw", etc.). Checking this requires comparing the first character with the n-th character, the second character with the (n-1)-th character, and so on. We can break down this problem using recursion. Let's see that in code.

Many technical interviews include programming problems that test your understanding of recursion. There are different resources for practicing for technical interviews, one that I hear about a lot is LeetCode. They have many recursion practice problems here.

3 Programming Problem: Prime Number Detection

My problem, formulated by ChatGPT!

You are tasked with creating a Java program that performs the following tasks:

1. **Generate Random Integers and Write to a File:**
   - Write a Java program that generates **200 random integers** between **1** and **1000** (inclusive).
   - Save these integers into a file named `numbers.dat`, with each integer on a new line.
2. **Implement Recursive Prime Checking:**
   - Write a recursive method named `isPrime(int number)` that determines whether a given integer is a prime number. The method should return `true` if the number is prime and `false` otherwise.
   - A prime number is a positive integer greater than **1** that has no positive integer divisors other than **1** and itself.
3. **Read from File and Find Primes:**
   - Read the integers from `numbers.dat`.
   - Use your recursive method to check each integer to determine if it is a prime number.
4. **Write Prime Numbers to a New File:**
   - Write all prime numbers found in the previous step to a new file named `prime.txt`, with each prime number on a new line.

Instructions

- You may use any file I/O approach (e.g., `FileOutputStream`, `FileWriter`, `BufferedReader`, `BufferedWriter`) that you are comfortable with.
- Ensure your recursive method for checking prime numbers is efficient.
- Handle exceptions, such as `IOException`, gracefully in your code.

Deliverables

Your solution should include the following methods:

1. `public static void generateRandomNumbers(String filename, int count, int min, int max)`
   - Generates **count** random integers between **min** and **max** (inclusive) and writes them to the specified `filename`.
2. `public static boolean isPrime(int number)`
   - A recursive method that returns `true` if the `number` is a prime number and `false` otherwise.
3. `public static void findPrimesAndWriteToFile(String inputFile, String outputFile)`
   - Reads integers from `inputFile`, checks each integer to determine if it is prime using the `isPrime` method, and writes all prime numbers to `outputFile`.

Example

File: `numbers.dat` (example content):

```
123
7
250
31
499
```

File: `prime.txt` (expected content if the numbers 7, 31, and 499 are primes):

```
7
31
499
```