Operating Systems and File Output

In this lecture we will cover:

- Basic operating systems and file management
- Writing text files in java

Your future in CS

I used to include this on my slides, but since these slides have changed - going to just leave it up here for every notebook. I get a lot of questions about more programming courses, the concentrations, and minors in computer science. Here is a brief reminder.

CS 165 – Next Course In Sequence, also consider CS 220 (math and stats especially)

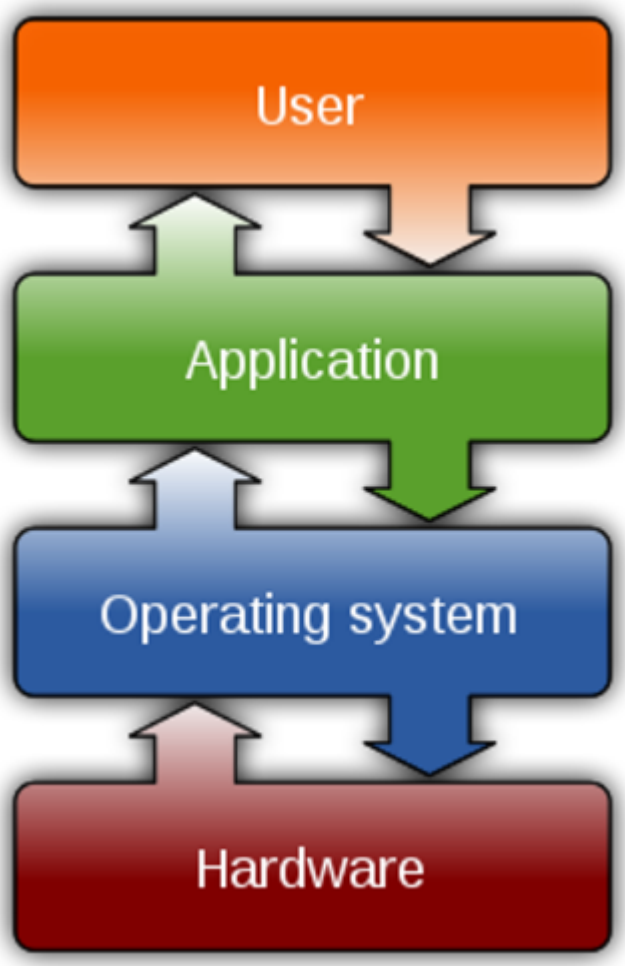
- CO Jobs Report 2021 – 77% of *all* new jobs in Colorado require programming
- 60% of all STEM jobs requires *advanced* (200-300 level)
- 31% of all Bachelor of Arts degree titled jobs also required coding skills
- 2016 Report found on average jobs that require coding skills paid $22,000 more

- Concentrations in CS:
    - Computer science has a number of concentrations.
        - General concentration is the most flexible, and even allows students to double major or minor pretty easily.
        - Software Engineering
        - Computing Systems
        - Human Centered Computing
        - Networks and Security
        - Artificial Intelligence
        - Computer Science Education.
    - Minors:
        - Minor in Computer Science - choose your own adventure minor
        - Minor in Machine Learning - popular with stats/math, and engineering
        - Minor in Bioinformatics - Biology + Computer Science

Operating Systems

- You use them daily
    - Most common OS in the world?

    - Android
        - written in java w/ Kotlin
- Operating Systems Control
    - Resources
    - Hardware Information
    - Devices
    - Running Applications, memory, etc
    - **File Storage!**

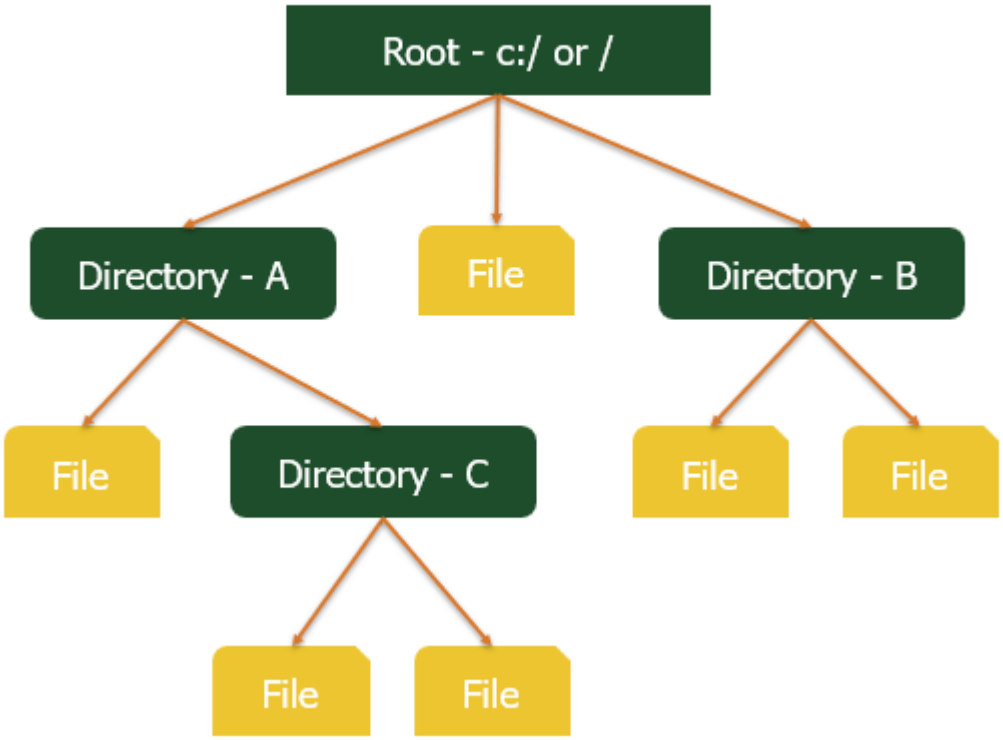


File Storage

Files is how we store information about programs. Without the ability to save files, we wouldn't have 'state' between applications runs.

Most operating systems are based on an "inverted tree", with the 'root' being at the top.

- Relative Path
    - relative to the current location
        - Examples:
            - "file"
            - "./file"



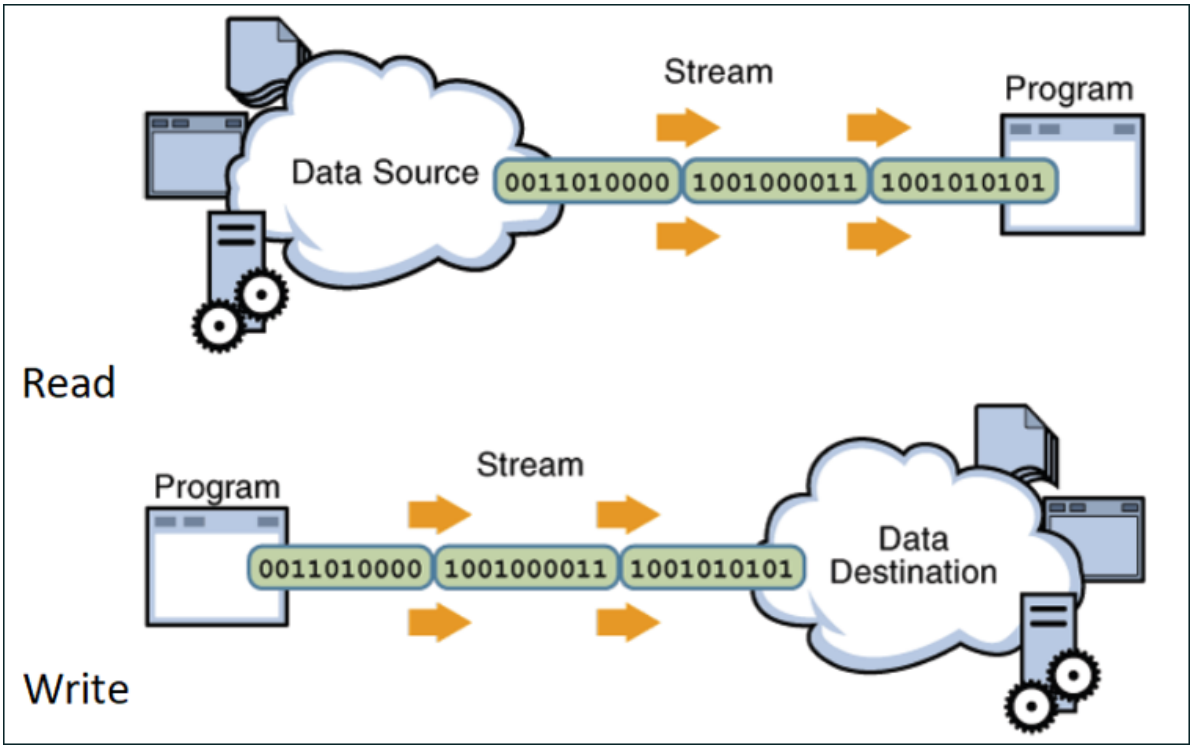
        - "../file"
        - "directory/file"
    - Shortcuts:
        - . - shortcut for 'current' directory
        - .. - shortcut for directory above this directory
        - notice the file names are just strings!
- Absolute Path
    - relative to the top of the hierarchy
    - Windows: drive letter with C:/, D:/ etc
    - Linux/MacOS/Unix: just a "/"
    - Examples:
        - "/directory/file"
        - "/directory/directory/file"
        - "C:\directory\file"
        - "/file"

NOTE: In the prerecorded videos there is a mini-tutorial on browsing files via the command line. You should try that on your own!

Files and Streams

Reminder: reading a file is coming in from the "input stream".

Writing a file is the "output stream".



To read a file we use:

```
Scanner in = new Scanner(new File("input.txt"));
```

To write a file we use:

```
PrintWriter writer = new PrintWriter(new File("output.txt"));
```

We can also use FileInputStream or FileOutputStream respectively.

PrintWriter

- The PrintWriter class is designed to 'print/write' to Output Streams.
- What are some Output Streams?
  - System.out - stream to the console
  - System.err - stream to the error log (often console)
  - File - can be an output stream
  - FileOutputStream - yes, an output Stream
  - Technically others!
    - Anything that extends the java OutputStream class
- PrintWriter has many of the same methods as System.out (intentionally)
  - `.println()`, `.print()`, . `printf()`
  - requires `.close()` to be called before anything is sent to the Stream!

In [1]:
```
/// So since it takes any *output* stream, lets try it with System.out
import java.util.Random;

Random rnd = new Random();

PrintWriter sysout = new PrintWriter(System.out);
```

```
for(int i = 0; i < 10; i++) {
    sysout.println(rnd.nextInt(10));
}
```

Wait nothing printed?!?

In [2]:
```
sysout.close();  // must close PrintWriter() to get it print!
```

```
7
4
6
4
5
5
6
7
2
4
```

Let's add files!

- However, with a file, we need to add a try/catch statement.
- Why?
  - Because what if the operating system rejects your access?
  - We will explore this in more detail, but that is what is meant by try/catch

In [5]:
```
public static String getClientInput() {
    Scanner scanner = new Scanner(System.in);
    String line = "";
    String fullText = "";
    System.out.println("Enter lines, and it will continue to loop until you type \"enc
    do {
        fullText += line + "\n";
        line = scanner.nextLine();


    }while(!line.equalsIgnoreCase("end"));
    return fullText;
}

try {
    PrintWriter writer = new PrintWriter(new FileOutputStream("data/notes.md"));
    // file extensions are just for the OS and humans, .md is still a text file full c
    writer.println("# My Notes!");
    writer.println(getClientInput());
    writer.close();
}catch(IOException ex) {
    System.out.println(ex.getMessage());
}
```

```
Enter lines, and it will continue to loop until you type "end".
```

Inclass Activity

This activity there is intentionally no 'starting files'. You will instead just create a file from scratch using your IDE.

You can write in your main method, or you can add additional methods, that is your call. I broke it up into two methods, one to read client input, and one to write out the information - just so I could focus on my tasks.

Your Task?

Write a program that takes in an input from the client which is only a number of random numbers to generate. You will then generate a number of random numbers equal to that input, and write each random number generated *out* to a **file** called `output.txt`.

Breaking down your tasks

- First, make sure you can read input from a client!
- Second, make sure you can generate random numbers based on the client's input (see above)
- Third, just print the random numbers to the console
- Fourth, now work on printing those random numbers to the file!

This iterative cycle is necessary for working with programming, and breaking things down into parts.

In [14]:
```java
public class RandomWriter {
    private static final Scanner scn = new Scanner(System.in);
    private static final Random rnd = new Random();

    public static int readClientInput() {
        System.out.print("Enter in a number: ");
        while(! scn.hasNextInt()) {
            System.out.print("Invalid number, try again: ");
            scn.next(); // ignore input
            System.out.println();
        }
        int val = scn.nextInt();
        return val;
    }

    public static void printRandomNumbers(int count) {
        PrintWriter f;
        try {
            f = new PrintWriter(new FileOutputStream("data/output.txt"));

            for(int i = 0; i < count; i++) {
                int num = rnd.nextInt(10);
                System.out.println("TESTING: " + num);
                f.println(num);
            }
            f.close();

        }catch(IOException ex) {
            System.err.println("Error handling the file! " + ex.getMessage());
        }
    }

}
```

```java
int input = RandomWriter.readClientInput();
System.out.println("TESTING: " + input);
RandomWriter.printRandomNumbers(input);
```

```
Enter in a number: TESTING: 10
TESTING: 5
TESTING: 6
TESTING: 4
TESTING: 6
TESTING: 9
TESTING: 5
TESTING: 2
TESTING: 2
TESTING: 2
TESTING: 4
```

Overall

People make writing out to files harder than what it is!

If you can print to the console, you can write to a file. Just remember

- you are dealing with Strings (for this class)
- You should separate your concerns on out!