COP 3330, Spring 2013

File I/O

Instructor : Arup Ghosh
01-16-13

School of Electrical Engineering and Computer Science
University of Central Florida

File I/O

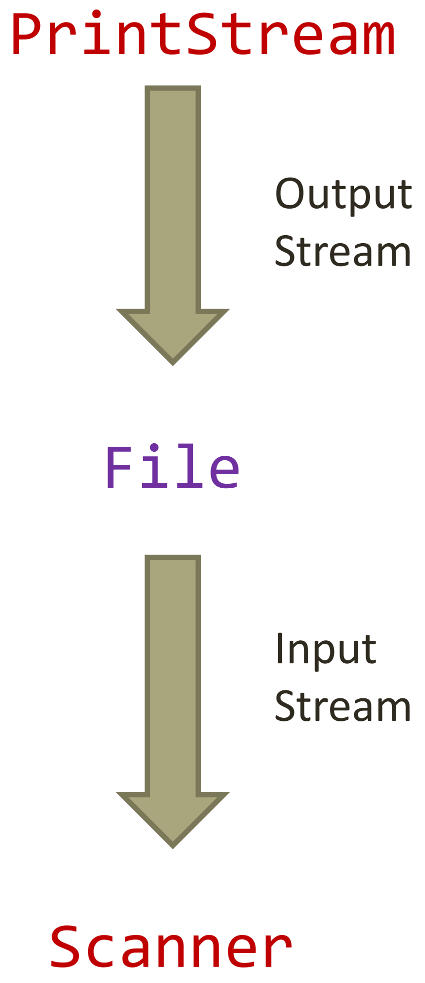
- With a few modifications, PrintStream and Scanner can be used to write to and read from a file.

- To represent a file, we use an object of a class called `File`.
  - The name kinda gives it away.

- We can hook up a PrintStream to a File, and do the same thing with a Scanner.

- Once that's done, all the methods behave exactly the same, except they now use the file instead of screen/keyboard.

File streams



- Write to the file by calling the print() methods of the PrintStream
- Read from it by calling the nextBlah() methods of Scanner.

File

- Creating a File object
  - Just provide a path.
  - `File myFile = new File(“C:\Users\arup\somefile.txt”);`
  - File lives in java.io, so import java.io.File or java.io.*;

- To read from the file:
  - `Scanner fileScanner = new Scanner(myFile);`

- To write to the file:
  - `PrintStream fileOut = new PrintStream(myFile);`

Absolute and Relative Paths

- Providing a complete unambiguous path (*absolute*) has a major drawback.

- If the program runs on someone else's computer, it may not have that directory structure.
  - Worse still if it runs on a different OS.

- However, we can provide paths *relative* to the directory our program runs in.
  - Command line: Whatever folder your .java file is in.
  - Eclipse/NetBeans: Both IDEs run your program from the project directory (not src or bin)

Relative Paths

- Most commonly, we just want to access files in the same directory as our program.
  - `File sameDirFile = new File(“blah.txt”);`

- In a subfolder of the program directory called foo (say):
  - `File fooDirFile = new File(“foo\blah.txt”);`
  - Note that Mac/Linux systems will use the forward slash instead.

- In the parent directory:
  - `File parentDirFile = new File(“..\blah.txt”);`
  - Basically, `..` is a pseudo-directory that refers to the directory containing the one you’re in.

Summary

- From Lastday's lecture:
- Java uses streams to handle I/O: `stdin, stdout, stderr`.

- `PrintStreams` are used to handle output, while `Scanners` can be used to handle input.

- From Today's lecture:
- By hooking them up to File objects, PrintStream and Scanner will handle file I/O as well.