

# Files

## Chapter 15

# Data

- Data in variables and arrays is temporary.
  - When your program ends, the data is lost.
- If you want to retain data beyond the running of your program, you need to use files (or a database, etc.).

# Streams

- Java uses *streams* to represent the source of input data or the destination for output data.
  - This includes when the source or destination is a file.
- The classes `InputStream` and `OutputStream` are used to read and write bytes (8-bit pieces of information).
  - This is pretty low-level.
  - Most files contain character data and so we should use more sophisticated streams.

# Streams

- Java uses *streams* to represent the source of input data.
  - This information is a file.
- The class `OutputStream` and write bytes (8-bit pieces of information).
  - This is pretty low-level.
  - Most files contain character data and so we should use more sophisticated streams.

## Important Note!

You might hear the term “streams” referring to the new additions made to Java 8. This is a very different idea than the simple I/O streams we are covering. (It’s also very cool stuff for those of you who want to explore!)

# Nesting Streams

- Some streams work with bytes in the file. Other streams work with characters. Others use buffering.
- You nest/combine these streams to get all of the functionality.

# File Output

- Use a `PrintWriter`, nested with a `BufferedWriter` and `FileWriter`
  - The `PrintWriter` class has `print` and `println` methods that work just like `System.out.print` and `System.out.println`

```
PrintWriter out =  
    new PrintWriter(  
        new BufferedWriter(  
            new FileWriter("filename.txt")));  
out.println("hello");  
out.close(); // CRITICAL! put in a finally  
             // block or use try-with-resources
```

# File Input

- Use a Scanner, nested with a FileReader

```
Scanner fileScan =  
    new Scanner(  
        new FileReader(  
            new File("filename.txt")));  
while(fileScan.hasNext()) {  
    String line = fileScan.nextLine();  
    System.out.println(line);  
}  
fileScan.close();
```

# Practice

- Read the contents of a file with one-word-per-line and write the contents out to a different file with all words on the same line (separated by a space).
  - For now, ignore exceptions.



# Parsing

- The `Scanner` class can be used to break down input based on white space or other delimiters that you choose.

```
String line = file.nextLine();
Scanner lineScan = new Scanner(line);
lineScan.useDelimiter(",");
while(lineScan.hasNext()) {
    String s = lineScan.next();
}
```

# Practice

- Review the user-data file created in Excel (and saved as a csv file). Write a program to read in the data and create objects from the data.

# **FILE I/O AND NIO**

# File Management

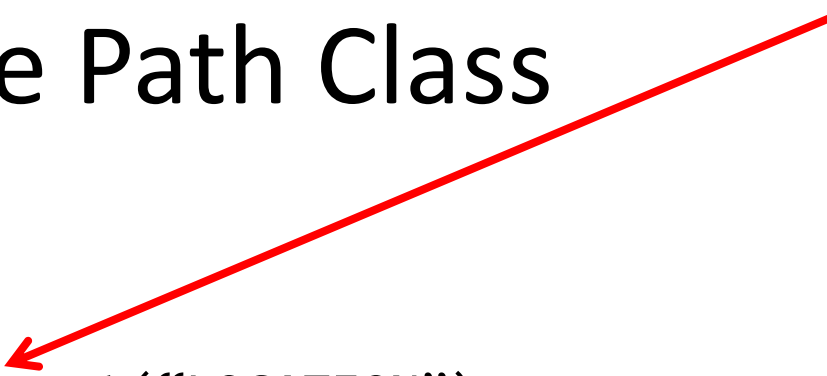
- We know how to read and write data from/to a file.
- Java also provides a way for us to manage files, including deleting, renaming, seeing when last modified, etc.
- Java SE 7 released new file handling classes as part of the `java.nio.file` package.

# The Path Class

- A Path is a sequence of directory names, optionally followed by a file name.
  - An absolute path is the full location of a file- from the root to the file
  - A relative path contains partial information

# The Path Class

helper  
class is  
Paths



- Retrieve a Path object

```
Path myPath = Paths.get("LOCATION");
```

- Location can be a single String or multiple Strings that each list a directory in the path (e.g., `Paths.get("home", "files", "java")`)
- **Methods**
  - `toString()`
  - `getFileName()`
  - `getName(int)`
  - `getNameCount()`
  - `getParent()`
  - `getRoot()`

# The Files Class

- Static methods for working with files and directories
- The Files methods work on Path objects
  - `Files.exists` and `Files.notExists`
    - Can return that the file exists, does not exist, or is unknown (the program does not have access)
  - `Files.isHidden`, `isReadable`, `isWritable`, **etc.**
  - `Files.isDirectory`
  - `Files.copy`
  - `Files.move`
  - `Files.delete`
  - `Files.createDirectory`
  - `Files.createFile`

# The Files Class

- You can also use `Files` to read/write from text files:

```
List<String> lines = Files.readAllLines(myPath);  
Files.write(newPath, lines);
```



# Practice

- Review the program that copies a user-specified file to a new directory.