# COMP 110/L Lecture 28

Kyle Dewey

## Outline

- Writing to files
- `finally`

# Writing to Files

–Same step as with reading files

# Writing to Files

Step 1: Create a `File` object

```
File myFile = new File(“myFile.txt”);
```

-Same step as with reading files

# Writing to Files

Step 1: Create a `File` object

```
File myFile = new File("myFile.txt");
```

Step 2: Create a `FileWriter` object

```
FileWriter fw = new FileWriter(myFile);
```

–Same step as with reading files

# Writing to Files

## Step 1: Create a `File` object

```
File myFile = new File("myFile.txt");
```

## Step 2: Create a `FileWriter` object

```
FileWriter fw = new FileWriter(myFile);
```

## Step 3: Create a `BufferedWriter` object

```
BufferedWriter bw =
  new BufferedWriter(fw);
```

–Same step as with reading files

## Writing to Files

Step 4: Write to `BufferedWriter` object as needed

```
bw.write(“Hello”);
bw.newLine();
bw.write(“World”);
bw.newLine();
```

–Same step as with reading files

# Writing to Files

## Step 4: Write to `BufferedWriter` object as needed

```
bw.write(“Hello”);
bw.newLine();
bw.write(“World”);
bw.newLine();
```

## Step 5: Close the `BufferedWriter` object

```
bw.close();
```

–Same step as with reading files

## Example:

`WriteStrings.java`

## `BufferedWriter`

Observation: `PrintWriter` seems to do everything `BufferedWriter` does, so why is `BufferedWriter` needed?

## `BufferedWriter`

**Observation: `PrintWriter` seems to do everything `BufferedWriter` does, so why is `BufferedWriter` needed?**

- Acts as a *buffer*
  - Layer between us saying `write` and the actual writing to the file
- Repeated short writes to files is **slow**
- Buffering idea: collect "writes" together in memory, then write to file all at once

-BufferedWriter transparently collects these writes in memory, and will write to the file when the space in memory is full.

finally

## Motivation

Sometimes we want to perform an action, whether or not an exception is thrown.

## Motivation

Sometimes we want to perform an action, whether or not an exception is thrown.

```
try {
  maybeThrowException();
  maybeDoThis();
} catch (SomeException e) {
  maybeDoThat();
} finally {
  alwaysDoThis();
}
maybeDoTheOtherThing();
```

-In the code above, the only thing guaranteed to always run is maybeThrowException (which might end early if it throws an exception), and alwaysDoThis.
-maybeDoThis will get skipped if maybeThrowException throws an exception
-maybeDoThat will get skipped if the body of the try does not throw a SomeException
-maybeDoTheOtherThing will get skipped if the body of the try throws an exception that isn't a SomeException, or if maybeDoThat throws an exception

## Example:
## `FinallyExample.java`

## Common Use

- `finally` is often used to make sure a file was closed, even if an exception was thrown while manipulating the file
    - `WriteStrings.java` will **not** do this
    - See `WriteStringsFinally.java`