FILE I/O

CS302 – Introduction to Programming
University of Wisconsin – Madison
Lecture 27

By Matthew Bernstein – matthewb@cs.wisc.edu

Introduction to File I/O

- What is File I/O?
  - It stand for File Input/Output – it is just the process of reading and writing files on your computer
- We will deal with reading from text files
- A text file is a file that simply contains characters
- Examples: .txt, .java, .html

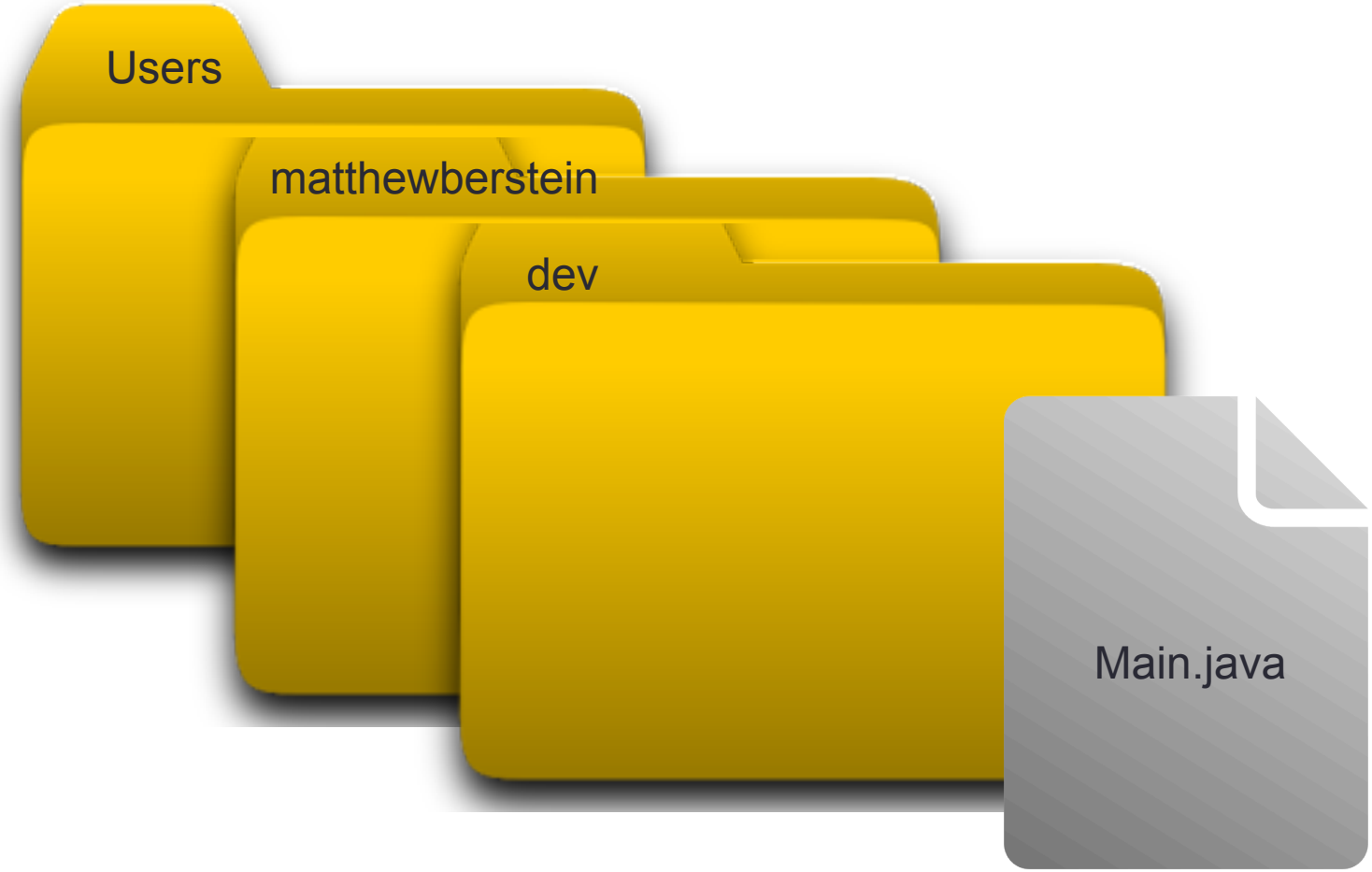
File Paths - UNIX

- On your computer, each of your file's location is defined by an address called the **absolute file path**
- On a UNIX system (such as Mac OSX or Linux), file paths look as follows:

/Users/matthewbernstein/dev/Main.java

UNIX File Paths

/Users/matthewbernstein/dev/Main.java



Windows File Paths

- On a Windows system file paths look as follows:

c:\Users\matthewbernstein\dev\Main.java

The root file is in the “c” drive

Using objects of the **File** class

- We use File objects for representing files on your computer
- Example:

```
// Create a file object that corresponds to a file
// named “myFile.txt” in a directory called
// “input”
File inputFile = new File(“/input/myFile.txt”);
```

Absolute Path

Using the **Scanner** for reading files

- We can use the Scanner for reading from files
- Remember:

```
Scanner scan = new Scanner(System.in);
```

Passing the 'System.in' object to the Scanner's constructor

- Now we pass the File object to the Scanner's constructor instead:

```
File inputFile = new File("/input/myFile.txt");
Scanner scan = new Scanner(inputFile);
```

What’s the Error?

```
Scanner scan = new Scanner(“input.txt”);
```

Windows or Unix?

- How can we write a program that will read a file using the file path pattern for ***either*** Windows or Unix-like systems?
- If we hardcode a file path to use the forward slash “/” then we are using the Unix file-path pattern
- If we hardcode a file path to use the backward slash “\” then we are using the Windows file-path pattern
- How do we fix this?

Use **File.separator**

- Each File object has a public static variable called **separator** that stores the string used to separate directories in a file path
- On a Windows system, **File.separator** will return "\"
- On a Unix-like system, **File.separator** will return "/"

- Example:

```
// path will store input/myFile.txt on Unix
//  and will store input\myFile.txt on Windows
String path = "input" + File.separator +  "myFile.txt";
```

Bonus Topic: Reading from a web page

- We have now seen how to pass a **System.in** object for reading input from the keyboard
- We have seen how to pass a **File** object for reading from a file on your computer
- You can pass the Scanner a **URL** object's **InputStream** for reading from a webpage

Writing to Files

- We write to a file using a **PrintWriter** object from the java.util package
- We pass a **File** object to the PrintWriter's constructor when creating a PrintWriter:

```
File outfile = new File("directory1/directory2/output.txt");
PrintWriter writer = new PrintWriter( outfile );
```

Create a File object corresponding
to the file on your computer that
you want to write to and pass this object
To the PrintWriter's constructor

Writing to Files

- We actually write text to a file by calling a FileWriter's **print** or **println** method
- Example:

```
PrintWriter writer = new PrintWriter(outFile);
writer.println("Hello World!");
```

- This will overwrite the content of the output file with "Hello World!"

Closing the Output Stream

- When your program is finished writing to the file you MUST call the PrintWriter's **close** method:

```
PrintWriter writer = new FileWriter( outFile );
writer.println("Hello World!");

writer.close(); // Close the output stream
```

- This method closes the output stream to the file
- If you don't close your PrintWriter, your program may terminate without correctly writing to the output file due to the fact that data may still be stuck in the PrintWriter's buffer
- Once you close the PrintWriter you can never use it again in your program. If you try to use it, you will get an **IOException**

IOException

- All of the your File I/O operations must be surrounded by a **try-block** followed by a **catch-block** to catch a possible **IOException**:

```
try
{
        // File I/O goes here
}
catch (IOException e)
{
        // Handle an exception here
}
```

Programming Exercises

- This idea is **challenging**:
- Write a Markov Model that will read text from a webpage and will generate random nonsensical text from the actual *text* on the page (not the html tags)
- Write a Markov Model that will generate random nonsensical sequences of html sections (example: image → paragraph → title → title)
- Combine the random text with the random html sections to generate a completely randomized web page

Cool CS Link of the Day

- http://www.youtube.com/watch?v=mmQl6VGvX-c
- Google's Knowledge Graph