DATA + FILE HANDLING

Data Structures + Algorithms
Dr. Monica Ortiz
Greenwich Academy

Overview

- Why Files?
- File Handling
  - *Scanner, a second look*
  - *Reading from files*
  - *Writing to files*

DATA IS USEFUL.

Data helps us make sense of the world around us. It helps us elucidate *trends* and can *make problems clear* so that we can solve them.

Data is stored in files

- To be useful, data must ultimately be stored
- .txt, .json, .jpg, .mp3, etc., depending on the type of information we want to store
- Generally, we use either *text* or *binary* files.
  - *We will use text only in this class*
- *Knowing how to bring this data into our code allows us to operate outside of a code vacuum.*
  - *to date, we have only dealt with code we wrote in its entirety without additional inputs (other than directly from our user)*

`Scanner`, a reminder

- What you have already learned:
  - `Scanner key = new Scanner(System.in);`
- Must `import java.util.Scanner;` at the top of the file to access the class
- `Scanner` allows you to access stuff from *outside* the current file: user input, file data, etc.

Recall…

```
import java.util.Scanner; // or .*
public class TestClass {
       public static void main(String[] args){
              Scanner key = new Scanner(System.in);
              System.out.print(“How old are you?  ”);
              int age = key.nextInt();

              System.out.println(“You are ” + age + “ years old.”);
       }
}
```

Create a `Scanner` object, use a print statement to get correct information from user, then print out something related to that information.

User `Scanner` to get information indirectly from the user

- Users can give us information through files
- What does a file look like to a computer?
  - *Need to use a `File` object*

FILES

`File` Objects

- [File class information]
- `File(String pathname)` is the constructor we'll use
  - *Example:* *`File f = new File("hamlet.txt");`*
- Remember that `File` is found within the `io` package of `java` subset, so you must `import java.io.File` (or `.*`) to use
- Some useful methods:
  - *`boolean canRead()`* *[tests whether file is readable]*
  - *`boolean exists()`* *[tests whether file and director exist]*

How do we use this within `Scanner`?

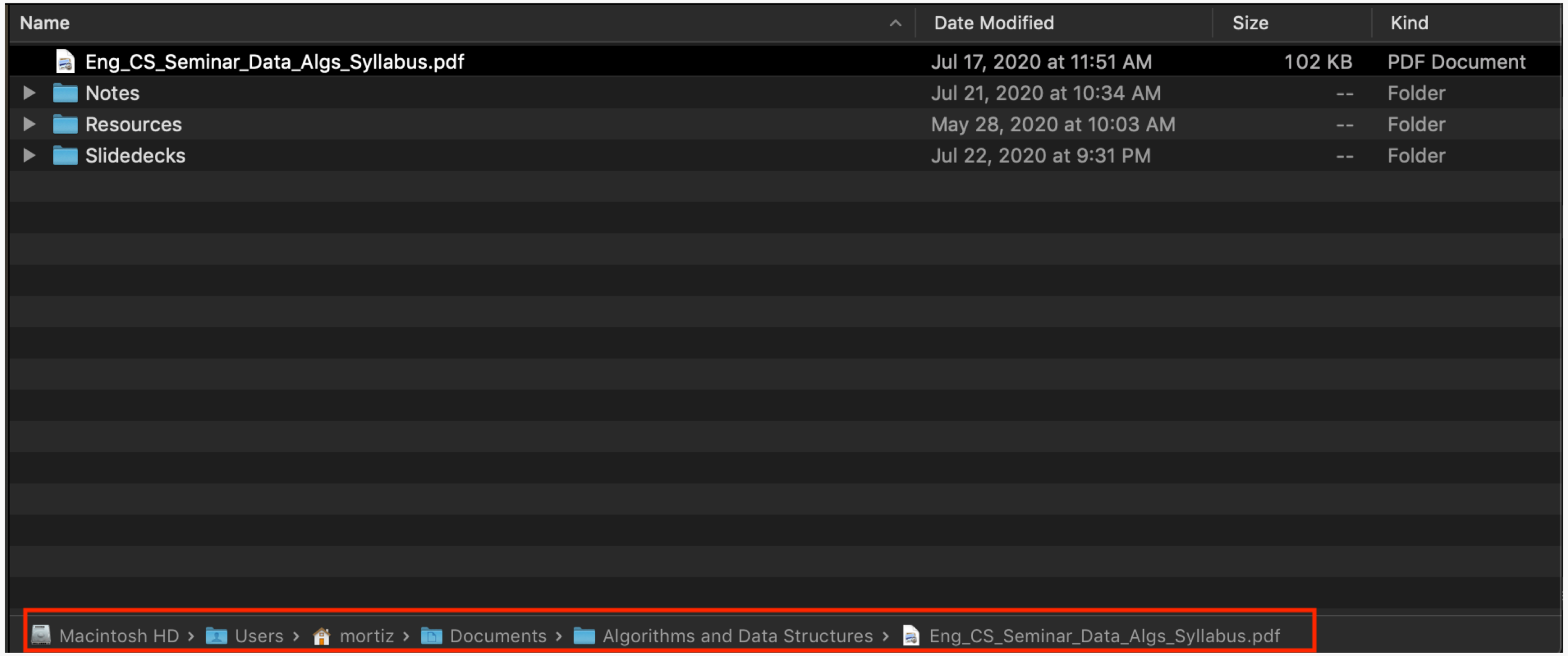
There is another constructor for `Scanner` that accepts a `File` object in place of `System.in`.

So, to create a `Scanner` that reads from a file, we use:

```
File f = new File(<path name as a String>);
Scanner s = new Scanner(f);
```

What is a “file path”?

A *file path* is nothing more than the system address of a file.



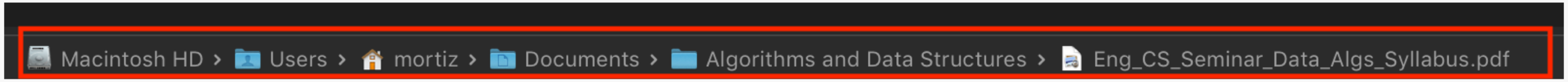
Name	Date Modified	Size	Kind
Eng_CS_Seminar_Data_Algs_Syllabus.pdf	Jul 17, 2020 at 11:51 AM	102 KB	PDF Document
Notes	Jul 21, 2020 at 10:34 AM	--	Folder
Resources	May 28, 2020 at 10:03 AM	--	Folder
Slidedecks	Jul 22, 2020 at 9:31 PM	--	Folder

Macintosh HD › Users › mortiz › Documents › Algorithms and Data Structures › Eng_CS_Seminar_Data_Algs_Syllabus.pdf

`"/Users/mortiz/Documents/Algorithms and Data Structures/Eng_CS_Seminar_Data_Algs_Syllabus.pdf”`

What is a “file path”?

The file path is the set of folders where you can find a specific resource



Macintosh HD › Users › mortiz › Documents › Algorithms and Data Structures › Eng_CS_Seminar_Data_Algs_Syllabus.pdf

So, for the file above, the *file path* would be:

```
/Users/mortiz/Documents/Algorithms and Data
Structures/Eng_CS_Seminar_Data_Algs_Syllabus.pdf
```

Note: spaces are evil. They make things more difficult. Do as I say, not as I do…

Let’s use a file in a program!

Here, we will write a program to read each line of a file and print it to the console.

Setup:

1. Open TextEdit and write some words. Use multiple lines.
2. Save your file to your Desktop as “textFile.txt”.
3. What is the file path for “textFile.txt”?
4. Now, create a new Java class and name it “WorkingWithFiles.java”.

Let’s use a file in a program!

5. Write a `public static void main` method that instantiates a `Scanner` using our file path.
   - Reminder: `File(String pathname)`
6. Next, we need to use our `Scanner` object to read each line using the `.hasNextLine()` and `.nextLine()` methods of the `Scanner` class.
   - Pseudocode: if the file has another line to read, read and print that line. Continue until there are no more lines to read.
7. Finally, it is customary that we `.close()` the `Scanner` after we are done, so go ahead and do that.
8. *RUN. Does it work?*

One more thing before this will run…

We don’t know if the file exists, which is why we get an error. So, we have to ensure that the program is robust enough to handle that error. One way to do that is using a `try-catch` statement.

```
public static void main(String[] args){
        try{
                File file = new File(“textFile.txt”);
                Scanner input = new Scanner(file);
                while(input.hasNextLine()){
                        String line = input.nextLine();
                        System.out.println(line);
                }
                input.close();
        } catch(Exception ex){
                ex.printStackTrace();
        }
}
```

Throwing Exceptions

- Now, there is an even easier way to accomplish this…
- The main thing to remember is that `File` objects throw a `FileNotFoundException`, which is a *checked exception*.
- A **checked exception** is an error that *must* be caught or declared in the header of the method that might generate it.
  - *Errrors occur at compile time*
  - *Unchecked exceptions are types of runtime errrors*
- **Throwing** an exception means that you are trying to handle a particular type of exception.

Throwing Exceptions

```
import java.io.*;
import java.util.Scanner;
public class Main{
        public static void main(String[] args)
            throws FileNotFoundException{
                Scanner input = new Scanner(new File(“textFile.txt”));
                while(input.hasNextLine()){
                        String line = input.nextLine();
                        System.out.println(line);
                }
                input.close();
        }
}
```

Note that you must use `java.io.*`, but need no `try-catch` statement.

How many numbers?

Create a list of numbers (integers are fine), one per line, and save them to a text file. Now, write a method, `public void int countValues(String pathname)`, that returns the number of values in the file as a print statement in the form:

```
count = 34
```

Calculate and report the average

Create a list of numbers (integers are fine), one per line, and save them to a text file. Now, write a method, `public static double calcAverage(String pathname)`, that calculates and returns the average of the values in the file as a `double`.

Calculate and report an array of averages

Create another file where each of the lines represents the set of numbers (again, integers are fine) to be averaged together. Write a method, `public static List<Double> calcAverages(String pathname)`, that calculates and returns the average of each set of numbers as an `ArrayList` of `Double` values. Why wouldn’t I suggest using an array for this?

You may want to examine another method in `Scanner`, the `.next()` method, which will help you parse a single line.

Report Workers' Hours

Using these two files, Employee.java and workHours.txt, create a method that the method, `public List<Employee> calcEmployeeHours(String pathname)`, should parse the text file for each employee, create an `Employee` object for each, and return an `ArrayList` of `Employee` objects.

Count coins

Write a method called `countCoins` that accepts a Scanner representing an input file whose data is a series of pairs of tokens, where each pair begins with an integer and is followed by the type of coin, which will be “pennies”, “nickels”, “dimes”, or “quarters”. Add up the cash value of all the coins and print the total money.

e.g., `3 pennies 2 quarters 1 pennies 23 nickels 4 dimes`

prints

```
Total money: $2.09
```

e.g., `3 pennies 1 nickels 1 pennies`

prints

```
Total money: $0.09
```

SCANNER TO READ
PRINTSTREAM TO WRITE

`PrintStream` is for writing to files

- [PrintStream class information]
- `PrintStream p = new PrintStream(File file);`
  - *This method requires that the file already exists. You are simply creating a Java object to access said file.*
- `PrintStream p = new PrintStream(String pathname);`
  - *This method allows you to create a filename(if it doesn't yet exist) and access it for writing in one step. Will overwrite prior data if you aren't careful!*
- How to import the `PrintStream` class correctly?

Let's think about `System.out.println()`

- The `println` method is used on the `System.out` object
- `System.out` prints to the terminal
- Here, we want to print to a *file* instead of to the terminal
  - *Overall, it's pretty similar!*

Back to `PrintStream`…

- `PrintStream p = new PrintStream(String pathname);`
- `p.println(“hello, file.”); // works just like you think`
- `p.print(“Here’s a second line.”); //this, too!`

Reverse the lines

- Write a method, `public static void reverseTheLines(File f)`, that looks at each line of a file and writes the reverse of that line to another file.
- e.g.,

  ```
  this looks
  a bit weird
  ```

  writes:

  ```
  skool siht
  driew tib a
  ```

Reformat names

- Accept a text file that contains names and birthdays, each person on a single line.
- The file should have lines of the following format:

```
    Schneider, Adrian 11/11/2015
```

- Print out a file that has each line reformatted as follows:

```
    Adrian Schneider: November 11, 2015
```

Print Duplicates

- Write a method called `printDuplicates` that takes as a parameter a `Scanner` containing a series of lines. The method should look for consecutive occurrences of the same word and print these to a file.

- e.g.,

```
hello hello how are are you you
I I i would love love love to get get get get lunch
```

  should print:

```
hello*2 are*2 you*2
I*2 love*3 get*4
```