Text Files

Additional ways to use Scanner

1

Text Files vs. Binary Files

- Text files store data as a sequence of binary character codes.
  - Text files can be read by standard editors.
  - TXT, HTML, PS, JAVA
- Binary files store data in a raw format where the binary data is not treated as characters.
  - Images: GIF, JPG, BMP
  - Audio: MP3, WAV
  - Video: MOV, AVI
  - Documents: DOC, WP, PDF, XLS

2

Text Files in Java

- Reading from a text file is similar to reading from the keyboard.

```
Scanner scan =
          new Scanner(new File(nameOfFile));
System.out.println("Reading from file...");
String fileInput = scan.nextLine();
```

3

Text Files in Java

- Writing to a text file is similar to displaying to the screen.

```
PrintWriter outfile = new PrintWriter(
  new FileWriter(nameOfFile));
System.out.println("Writing to file...");
outfile.print(outputText);
outfile.println(outputText);
outfile.close();
```

4

`IOException`

- Opening up a text file for reading can cause an `IOException` to be thrown if the file cannot be found.
- Opening up a text file for writing can cause an `IOException` to be thrown if there is a problem with the file system so a file cannot be created (out of space, etc.)
- More about exceptions later this semester.

5

Initialization

REQUIRED IMPORTS:

```
import java.util.*;
import java.io.*;
```

```
public static void main(String[] args)
  throws IOException {
     Scanner scan = new Scanner(
               new File("data.txt"));
     PrintWriter outfile = new PrintWriter(
               new FileWriter("results.txt");

     // YOUR CODE GOES HERE
  }
}
```

6

Example: Line Numbering

```
public static void main(String[] args)
  throws IOException {
     Scanner scan = new Scanner(
          new File("data.txt"));
     PrintWriter outfile = new PrintWriter(
          new FileWriter("results.txt"));

     String fileInput;
     int lineNum = 0;
```

Example: Line Numbering
(cont'd)

```
     while (scan.hasNextLine()) {
          fileInput = scan.nextLine();
          lineNum++;
          outfile.println(lineNum + ": " +
               fileInput);
     }
     outfile.close();
  }
}
```

Example:
Initializing an array from a text file

```
8 ◄──────── first entry indicates the number of data values
            in the file (not including this value)
19
53
25
77
34
-67
153                              data.txt
2
```

Example:
Initializing an array from a text file

```
public static void main(String[] args)
  throws IOException {
     Scanner scan = new Scanner(
          new File("data.txt"));
     int numValues = scan.nextInt();
     int[] dataArray = new int[numValues];
     for (int i = 0; i < numValues; i++)
          dataArray[i] = scan.nextInt();
     ...
```

Using `Scanner` in other ways

- Goal: We wish to add up all of the numbers listed in a file, but the file may have more than one number per line.
  - No arrays are used here.
- We can use one `Scanner` to read from the file.
- We can use another `Scanner` to take each line we read from the file and extract each number on that line one by one.

Example:
A more complex text file

define one Scanner to read each line from the file one at a time

```
48 23 53
19 13
53 932 324 53   ◄──
25 12 -133 4245 472
77
9 156 34
```

define another Scanner to read each integer from the line, one at a time

nums.txt

(first entry does NOT indicates the number of data values in the file!!!)

Using `Scanner` in other ways

```
public static void main(String[] args)
  throws IOException {
    Scanner filescan = new Scanner(new File("nums.txt"));
    int sum = 0;
    while (filescan.hasNextLine()) {
         String line = filescan.nextLine();
         Scanner linescan = new Scanner(line);
         while (linescan.hasNextInt()) {
              sum += linescan.nextInt();
         }
    }
    System.out.println("Total = " + sum);
}
```

13