



# CS0007 Recitation

THURSDAYS 12:00-12:50PM

MICHAEL BARTLETT



# Overview

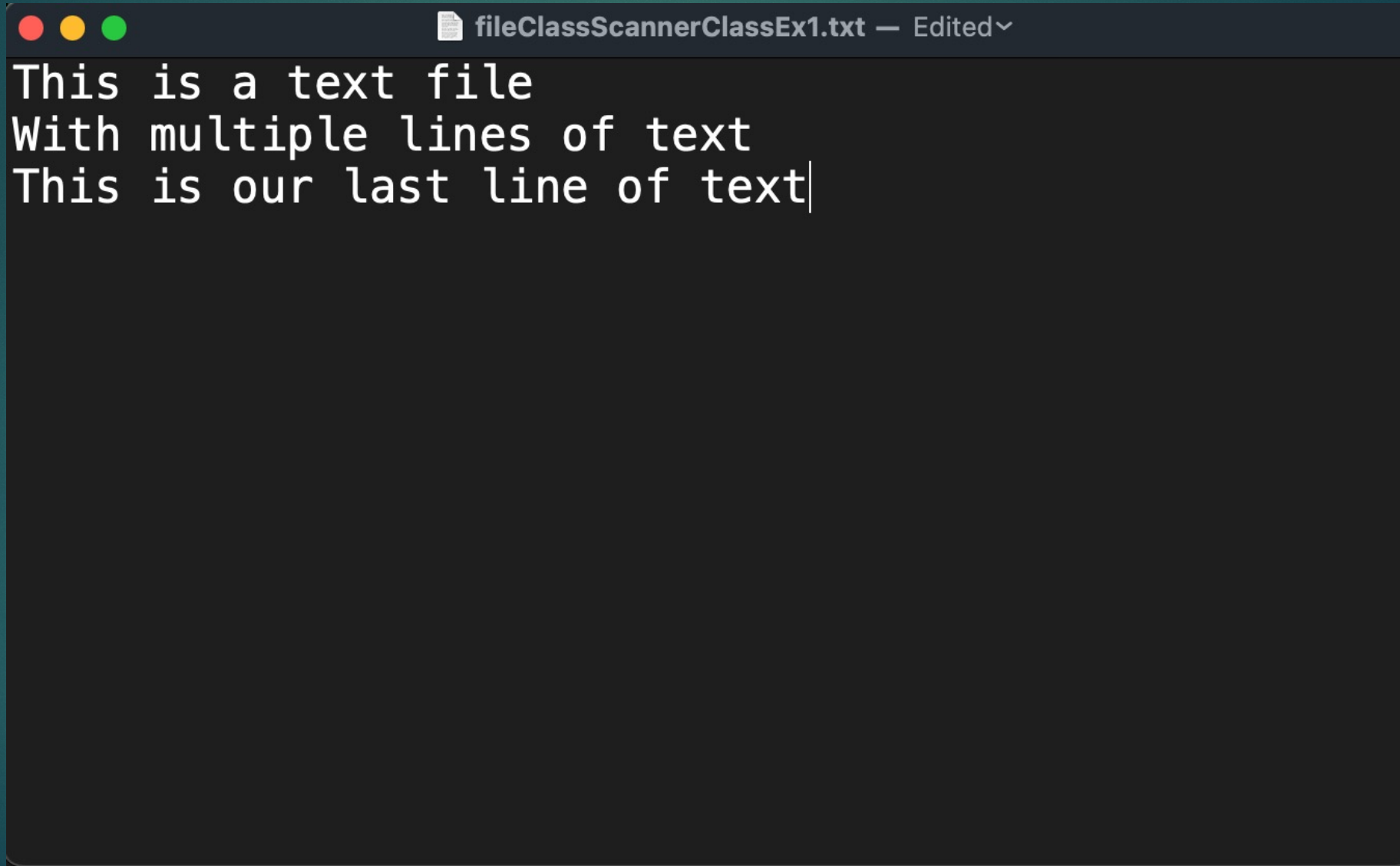
- ▶ Reading Text Files
- ▶ Writing Text Files
- ▶ Exceptions
- ▶ Arrays
- ▶ Lab 6



# Reading Text Files Using Scanner

- ▶ First, we need to use the File Class to get a reference to the file that we want to read.
- ▶ Then we use a Scanner to read the file and do something with the data.
- ▶ Scanner has multiple methods that we can utilize to read the file.
  - ▶ [Check out the Java API docs for Scanner](#)

# Reading Text Files Using Scanner



```
fileClassScannerClassEx1.txt — Edited✓  
This is a text file  
With multiple lines of text  
This is our last line of text|
```



# Reading Text Files Using Scanner

```
File file = new File("fileClassScannerClassEx1.txt");
Scanner sc = new Scanner(file);

while (sc.hasNextLine()) // Checks if the file has another line
{
    System.out.println(sc.nextLine()); // Prints the next line while the condition is true
}
```

→ This is a text file  
With multiple lines of text  
This is our last line of text

```
File file1 = new File("fileClassScannerClassEx1.txt");
Scanner sc1 = new Scanner(file1);

while(sc1.hasNext()) // Checks if file has another "token"
{
    // In this case a "token" is a string of characters that is terminated by a space
    System.out.println(sc1.next()); // Prints the next word
}
```

→ This  
is  
a  
text  
file  
With  
multiple  
lines  
of  
text  
This  
is  
our  
last  
line  
of  
text



# Reading Text Files Using BufferedReader

- ▶ First, we need to use the File Class to get a reference to the file that we want to read.
- ▶ Then we use a BufferedReader to read the file.
- ▶ Scanner has multiple methods that we can utilize to read the file.
  - ▶ [Check out the Java API docs for BufferedReader](#)



# Reading Text Files Using BufferedReader

```
public static void main(String[] args) throws Exception {  
    File file = new File("fileClassScannerClassEx1.txt");  
    BufferedReader br = new BufferedReader(new FileReader(file)); // Declare new buffered reader while using FileReader to read the file in  
    String str; // String to hold values to print  
  
    System.out.println();  
    System.out.println();  
  
    while ((str = br.readLine()) != null) // Checks if the file has another line  
    {  
        System.out.println(str); // Prints the next line while the condition is true  
    }  
  
    System.out.println();  
    System.out.println();  
}
```



This is a text file  
With multiple lines of text  
This is our last line of text



# Scanner Vs. BufferedReader

- ▶ Scanner will *parse* every “token” in a file that way you can interpret all the “tokens” in a file
  - ▶ Parsing: Interpreting the given input as tokens (parts). It's able to give back you specific parts directly as int, string, decimal, etc. See also all those nextXx() methods in Scanner class.
- ▶ BufferedReader will simply read the stream without doing any special parsing
  - ▶ Reading = Dumb streaming. It keeps giving back you all characters, which you in turn have to manually inspect if you'd like to match or compose something useful. But if you don't need to do that anyway, then reading is sufficient.



# Writing to files Using PrintWriter

- ▶ First, make a new PrintWriter object and set the file name.
- ▶ Then, write your data.
- ▶ Make sure to close it using `writer.close()`;
  - ▶ Your data won't write if you don't do this.

# Writing Text Files Using PrintWriter

```
public class PrintWriterEx {  
    Run | Debug  
    public static void main(String[] args) throws Exception {  
        PrintWriter outputFile = new PrintWriter("output.txt");  
  
        for (int i = 0; i < 25; i++)  
        {  
            outputFile.println("This is line " + i);  
        }  
        outputFile.close();  
    }  
}
```



```
output.txt ~  
This is line 0  
This is line 1  
This is line 2  
This is line 3  
This is line 4  
This is line 5  
This is line 6  
This is line 7  
This is line 8  
This is line 9  
This is line 10  
This is line 11  
This is line 12  
This is line 13  
This is line 14  
This is line 15  
This is line 16  
This is line 17  
This is line 18  
This is line 19  
This is line 20  
This is line 21  
This is line 22  
This is line 23  
This is line 24
```



# Appending Data To Existing File Using FileWriter

- ▶ First, make a new File object
- ▶ Then, make a new FileWriter object and set the file name.
- ▶ Then, make a BufferedWriter object out of your FileWriter object
- ▶ Write your data and then close the BufferedWriter and FileWriter objects.

# Appending Data To Existing File Using FileWriter



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

public class FileWriterEx {
    Run | Debug
    public static void main(String[] args) throws IOException {
        File file = new File("output.txt");
        FileWriter fw = new FileWriter(file, true); // take in the file name and set to true (looking at the Java API the boolean value is to append or to overwrite the data.)
        BufferedWriter bw = new BufferedWriter(fw);
        bw.write("line 25");

        bw.close();
        fw.close();
    }
}
```



```
output.txt - Edited
This is line 0
This is line 1
This is line 2
This is line 3
This is line 4
This is line 5
This is line 6
This is line 7
This is line 8
This is line 9
This is line 10
This is line 11
This is line 12
This is line 13
This is line 14
This is line 15
This is line 16
This is line 17
This is line 18
This is line 19
This is line 20
This is line 21
This is line 22
This is line 23
This is line 24
line 25
```



# Handling Exceptions

- ▶ Exceptions happen when we can't 100% verify that the code will work while being compiled.
- ▶ A good example of an exception would be trying to open a file that doesn't exist.
  - ▶ If your program can't find a file that you're trying to open, a `FileNotFoundException` gets thrown by Java.
- ▶ You could let Java know that your method may throw the exception (as seen in the previous examples)
- ▶ Or, you could use a try catch block to handle an exception that may be thrown
  - ▶ You can then decide what happens if an exception happens instead of Java just stopping your program.



# Handling Exceptions



```
public class fileNotFoundEx {  
    public static void main(String[] args) throws FileNotFoundException{  
        try {  
            InputStream inFile = new FileInputStream("fileThatDoesn'tExist.txt");  
        } catch (FileNotFoundException e) {  
            System.out.println("File not found. Program Exiting.");  
        }  
    }  
}
```



```
michaelbartlett@Michaels-MacBook-Pro-2 Examples % java fileNotFoundEx  
File not found. Program Exiting.  
michaelbartlett@Michaels-MacBook-Pro-2 Examples %
```



# Arrays

- ▶ Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- ▶ When making an array, declare the type that's going to be stored in it and name it. Then set the values that you want in the array.
- ▶ Or you can declare an array that's not equal to anything and just has its size in the square brackets.

# Creating Arrays

```
public class arrayEx1 {  
    Run | Debug  
    public static void main(String[] args) {  
        String[] cars = new String[] {"Volvo", "BMW", "Ford", "Mazda"}; // Make a new array with its values right away  
  
        for (int i = 0; i < cars.length; i++) // Loop through the array and print every element  
            System.out.println(cars[i]);  
  
        int[] nums = new int [10]; // Allocate room for 10 elements of an array in memory  
  
        for (int i = 0; i < nums.length; i++) // Fill the array with the number that their space in the array represents  
            nums[i] = i;  
  
        for (int i : nums) // "For each (int) i in nums print i"  
            System.out.println(i);  
    }  
}
```



```
michaelbartlett@Michaels-MacBook-Pro-2 Examples % java arrayEx1  
Volvo  
BMW  
Ford  
Mazda  
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```



# Multidimensional Arrays

```
public static void main(String[] args) {  
    // declaring and initializing 2D array  
    int arr[][] = { {2,7,9},{3,6,1},{7,4,2} };  
  
    // printing 2D array  
    for (int i=0; i< 3 ; i++)  
    {  
        for (int j=0; j < 3 ; j++)  
            System.out.print(arr[i][j] + " ");  
        System.out.println();  
    }  
}
```



```
michaelbartlett@Michaels-MacBook-Pro-2 Examples % java multiDimensionalArrayEx  
2 7 9  
3 6 1  
7 4 2
```

# ArrayLists in Java

- ▶ ArrayLists provide us with dynamic arrays in Java.
- ▶ Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.



# Multidimensional Arrays

```
public static void main(String[] args)
{
    // Size of the
    // ArrayList
    int n = 5;

    // Declaring the ArrayList with
    // initial size n
    ArrayList<Integer> arrli
        = new ArrayList<Integer>(n);

    // Appending new elements at
    // the end of the list
    for (int i = 1; i <= n; i++)
        arrli.add(i);

    // Printing elements
    System.out.println(arrli);

    // Remove element at index 3
    arrli.remove(3);

    // Displaying the ArrayList
    // after deletion
    System.out.println(arrli);

    // Printing elements one by one
    for (int i = 0; i < arrli.size(); i++)
        System.out.print(arrli.get(i) + " ");

    System.out.println();
}
```



```
michaelbartlett@Michaels-MacBook-Pro-2 Examples % java ArrayListExample
[1, 2, 3, 4, 5]
[1, 2, 3, 5]
1 2 3 5
```



# Arrays vs. ArrayLists

## The Similarities:

- ▶ Array and ArrayList both are used for storing elements.
- ▶ Array and ArrayList both can store null values.
- ▶ They can have duplicate values.
- ▶ They do not preserve the order of elements.



# Arrays vs. ArrayLists

## Key Differences:

- ▶ Arrays are static in size, meaning that if you need to upsize it, it has to be copied into a new array before you can add more elements.
  - ▶ You need to know the size of the array when declaring it.
  - ▶ This makes it faster though
- ▶ ArrayLists are dynamic in size, meaning that you don't need to make a new one to upsize it.
  - ▶ The size isn't necessary when declaring it.
  - ▶ ArrayList is internally backed by the array in Java so the resize operation slows down the performance
- ▶ Arrays can be multi-dimensional whereas ArrayLists can only be single dimensional.



# Lab 6

- ▶ Check out Paulo's videos for the explanations
- ▶ Key Concepts:
  - ▶ Reading data from a textfile into an arraylist
  - ▶ Access arraylist elements based on user input
  - ▶ Validating user input



# For next week

- ▶ It looks like y'all will have an exam coming up so we'll talk about that and any other concepts you learned that week
- ▶ This lab will take awhile and is a lot more than you've done before.
  - ▶ Do it in small pieces and test as you go!
- ▶ **Please email me or if you're having issues with submitting labs on time**
  - ▶ **Or if you're having any issues with the labs/quizzes! If you don't understand how to do something just ask!**
- ▶ We made a Discord server! <https://discord.gg/23weGMFk>
  - ▶ Joining is optional, but it'll be a good point of contact with us