# Lab04a – ArrayList – Datafiles and try/catch

## Catch a mistyped file name

* As you know, it is easy for a user mistype a file name.
* If that happens, the File object will search the hard drive in vain

throw a FileNotFoundException.

Example: The try/catch code below catches the exception, prints a message, and ends the program.

When you use a try-catch, you don't need **throws Exception** here.

**public class** Driver

{

**public static void** main(String[] args)

{

Scanner infile = **null**;

**try**

{

String filename = JOptionPane.showInputDialog("Enter filename");

infile = **new** Scanner( **new** File(filename) );

}

**catch**(**FileNotFoundException** e)

{

JOptionPane.showMessageDialog(**null**, "Error: File not found.");

System.exit(0);  
 }

* Methods that could throw an exception should be in the try block.
* The attempt to recover should be in the catch block, which is only executed if needed.

NOTE: that the catch block shown above does not give the user a second chance.

* to Import
  + Data.txt

6525  
390.0202356936466  
948.1569277972972  
638.3505188754447  
574.3344213810659  
 .

.

* + - The first line contains an integer 6525, which tells the number of numbers in the file.
* to create
  + Driver04.java – (use Driver03.java as a template)
    - Create an ArrayList of Doubles
    - Then read the integer, create an ArrayList, and read the rest of the data into the ArrayList.
      * (In practical terms, the line public static final int NUMITEMS = 6525; does not exist in Lab04 because NUMITEMS will change for each data file.)
    - Finally, process the ArrayListg for sum, average, max, and min.
    - Use data.txt. This is the data file.
    - After completing the above:
      * *Put a loop around the try/catch block. Keep prompting the user to enter a filename until an existing infile is found.*
      * Modify the program again so that the user is given only three chances to enter a good filename. If the three chances are used up, display an appropriate message and terminate the program.
      * Import and Compile and run MakeDataFile.java.
        + Verify that the set of data in "data.txt" is different.
        + Run Driver04.java again.
        + How can the same driver process different sets of data?

Reading Text Files

# Text Files

* Used for large data
* Data comes from a file previously stored on your hard drive.

## Example: Driver00

* Written to read the data from a text file.
* Reading from a text file requires four new statements.
  + The prompt is no longer needed.
  + Changes are highlighted

1 **import java.io.\*;**  //for File

2 **import java.util.\*;**  //for Scanner

3 **public class** Driver00

4 {

5 **public static final int** NUMITEMS = 10;  
6 **public static void** main(String[] args) **throws Exception**

7 {

8 **double**[]array **= new** **double**[NUMITEMS];

9 Scanner infile **= new** Scanner( **new** File("filename.txt") );

10 **for**(**int** x = 0; x < NUMITEMS; x++)

{

11 array[x] = infile.nextDouble();

}

12 infile.close();

13 System.out.println("The numbers in the file, backwards:");

14 **for**(**int** x = 0; x < NUMITEMS; x++)

15 System.out.print("\t" + array[NUMITEMS – x – 1]);

16 }

17 }

* + Lines 1 and 2:
    - package java.io contains the File class,
    - java.util contains the Scanner class.
  + Line 6:
    - throws Exception, required because we have new File on Line 9, discussed later
  + Line 9:
    - The Scanner object accepts a File object, whose argument, in quotation marks, is the name of the text file.
    - We don't have to know exactly how these objects work!
  + Line 11:
    - If you instantiate the Scanner object as in Line 9, then infile.nextDouble() takes care of the details of reading the text file.
    - It even converts (if it can) the string data into double data.
    - If the string can't be converted, the method will *throw an exception*, which stops the program.
    - We discuss exceptions later.
  + Line 12:
    - At this point, the data from the text file has been stored in the array.
    - Close the scanner object.

## Data in text file – stored as string

* It is the programmer's responsibility to know whether to change those strings into doubles or integers, or to leave them as strings.
* In the next labs, sometimes you will use
  + infile.nextDouble()
  + infile.nextInt()
  + infile.next().
    - The infile.next()method reads one *string*—
      * a sequence of characters up to whitespace—at a time from the file.
    - Loop: If you put infile.next()in a loop, then it reads string after string, one by one, straight though until the end is reached.
    - Error: If there is an error in reading, the program crashes. You can't read the same string twice. Neither can you read some, then go back and start over. If you need to access the same data twice, you can either use two scanners, or read the data into an array. That is why these labs usually read data into arrays.

NOTE:

* The name of a file as given on Line 9 is almost never "filename.txt"—that is a placeholder.
* Similarly, the name of the Scanner object on Line 9 does not have to be infile.

Exceptions

# Types of errors

## Errors caused by programmer:

* Syntax errors, run-time errors and logic errors.

The programmer is completely responsible for syntax and logic errors,

## Errors that stop execution – caused by user

* If the user typed the wrong password, or misspelled the file name.
* Still, the programmer would like to be able to handle these kinds of error gracefully, say by offering the user another chance.

## Recover gracefully - Java throws an Exception

* Usually an exception is thrown in one part of the program and caught in another part of the program

Example:

* In Lab03, when the method nextDouble encountered an input error,
* it threw an IOException to the main. In that lab, we decided not to deal with it there, but to pass it up to the system.
* We told the compiler we wanted to toss the error to the system by using throws Exception in the header:

**public static void** main(String[] args) **throws Exception**

## Catch it and deal with it gracefully

Java uses the try/catch block to do that. The basic structure is

**try  
{** //run some code which might throw an exception  
**}  
 catch (Exception e)  
 {**  
 //if thrown, try to deal gracefully with the exception

**}**

//the rest of the program

* Because all sorts of things can go wrong, Java provides many sorts of exceptions.
* Go to the API and look up the Exception class.
  + Then click on IOException and then on FileNotFoundException, which is the exception that we will be throwing in the next lab.
  + Running off the end of an array will throw an ArrayIndexOutOfBoundsException.
  + Incorrectly addressing an object throws a NullPointerException.
  + Improperly casting throws a ClassCastException.
  + Every time your program crashes, you are actually throwing a new RuntimeException, and so on.

## create your own exceptions

Example: such as DivideByZero, used like this:

**try  
 {**  
 //code to prompt the user to enter a denominator  
 **if** (denominator == 0)   
 **throw new** DivideByZero();  
 **}  
 catch** (DivideByZero e)  
 **{**   
 System.out.println("Sorry, you cannot divide by zero!");  
 System.exit(0);  
 **}**

//the rest of the program

**Lab04  
Exercises #1**

Below are the beginnings of three programming assignments. Read them carefully for all the information.

1. Your professor assigns you to write part of a program that processes a set of exam scores. She says there are exactly 181 exams. The scores are in a text file "exam01.txt", one score per line. Write the code fragment to read the exam scores into an array.

2. Up till now, your professor has kept exam scores on paper. She assigns you to write a program so that someone can enter the exam scores into an array, from the keyboard. The number of scores has to be entered from the keyboard as well. She wants you to use good, friendly prompts.

1. Your professor teaches several courses. Each set of exam scores is saved to disk under different filenames. Each set of exam scores has an integer in the first line, which tells the number of exam scores in the file. Write the code fragment that prompts the user for the file name, reads the first line, and creates an array of the correct size.

4. It turns out that, to calculate the sum, average, and max, you don’t need to store the data in an array. In this case you can process the data from the text file on the fly, without knowing in advance how much data you have. The Scanner class has two methods hasNext and nextDouble to help. Fill in the blanks:   
 Scanner infile = **new** Scanner( **new** File("data.txt") );  
 **double** sum = \_\_\_\_;   
 **double** max = Double.MIN\_VALUE;   
 **int** count = \_\_\_\_;  
 **while**( infile.hasNext() )  
 {  
 **double** x = infile.nextDouble();  
 sum = sum + \_\_\_\_\_;  
 max = Math.max(\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_);  
 count++;  
 }  
 System.out.println("Sum: " + \_\_\_\_\_\_\_);   
 System.out.println("Average: " + \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);  
 System.out.println("Max: " + \_\_\_\_\_\_\_);

Lab04  
Exercises #2

Study the code of MakeDataFile, shown below. In Extension 1 of Lab04, you used MakeDataFile to create a different data.txt. How does MakeDataFile work? Answer the questions below.

4 **import** java.io.\*;  
 5 **public** **class** MakeDataFile  
 6 {  
 7 **public static void** main(String[] args) **throws Exception**  
 8 {  
 9 PrintStream outfile = **new** PrintStream(   
 **new** FileOutputStream("data.txt"));

10   
11 **int** numitems = (**int**)(Math.random() \* 5000 + 5000);  
12 outfile.println(numitems);  
13   
14 for(**int** x = 0; x < numitems; x++)  
15 outfile.println(Math.random() \* 1000);  
16 outfile.close();  
17 }  
19 }

1. What does Line 9 do?
2. Why does Line 7 need throws Exception?
3. How can you catch the exception in a different way?
4. What does Line 11 do?
5. What does Line 12 do?
6. What do Lines 14 and 15 do?
7. Is Line 16 necessary? Y/N What does Line 16 do?
8. In the code above, outfile is a reference to a \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_object.
9. In the code above, the object referenced by outfile has been instantiated from class \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
10. If a "data.txt" file already exists on the hard drive, does **new** FileOutputStream("data.txt") overwrite that file or not?
11. In your own words, briefly describe what MakeDataFile does.