## Reading Data from Text Files

A file is an organized collection of information stored on a disk drive or some other storage device external to the computer's RAM memory.

The Scanner class makes it very easy to work with text files.

Instead of creating a Scanner object associated with **System.in**, or with a particular String object, we simply create one associated with a particular **File** object. The syntax is:

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

Here are some examples:

**Scanner reader = new Scanner( new File ("C:\\Data\\CD-Data.txt") ) ;**

**Scanner reader = new Scanner( new File ("A:Grades.dat") ) ;**

**Scanner reader = new Scanner( new File ( "Votes.data") ) ;**

- In the first example, Scanner object reader is associated with file *CD-Data.txt*, located in folder *Data* on the C: drive.

  ☞ Note that two backslashes are used instead of one (the backslash is how Windows indicates a folder). This is because the Java compiler interprets the backslash as the *escape character*. In Java, "\\" is the escape sequence for the backslash!

- In the second example, Scanner object reader is associated with file Grades.dat, located in the root directory of A:

- In the last example, no path is provided for the file Votes.data. If no path is provided, the default folder - the one containing the program - is assumed.

**If the specified file does not exist in the specified folder, or the folder does not exist in the specified path, a *FileNotFoundException* is thrown!**

☞ After creating the Scanner object, we read data from the file using the familiar methods next(), nextLine(), nextInt(), nextDouble(), and.hasNext()

## Imports and "Throws Clauses"

To work with data files, we need to import several classes that are in Java's "io" package. The easiest way to do this is:

**import java.io.* ;**



Reading from a data file presents a unique set of problems that does not exist when reading from the keyboard. What if we misspell the file name, or the file simply does not exit? Attempting to open a file will throw a **FileNotFoundException** if the file does not exist. (If we misspelled the file name, then, in fact, the file we are attempting to open does not exist). An **Exception** is an error in the Java language. Another **IOException** would be attempting to read past the end of the file. Because of the possibility of Exceptions being thrown, Java requires us to deal with the potential problem in either of two ways:

- Provide an exception handler (a special block of code that is executed when an exception occurs), or

- Tell the compiler that you are aware of the possibility of an exception and take responsibility should one occur, although you do not wish to "handle" it. This is done by appending a "throws clause" to the heading of a method that may throw an exception, or calls a method that may throw one.

Attempting to read from a data file after you have reached the end of the file will cause the Scanner class to throw a **NoSuchElementException**. This exception should also be caught and dealt with. It makes no sense to continue your program if you have failed to read in all necessary information. Since the NoSuchElementException class is in the Utility package, no other import statements are required.

## try/catch/finally (Outsource: 14.1 – 14.10)

Java has a robust, but complicated, exception handling framework. By using a **try** block, a developer can wrap a suspect block of code. If the code throws an exception, the accompanying **catch** block allows the developer to process, or handle, the exception. Handling the exception often involves logging it and then determining if the application can continue or if the application should exit.

An optional **finally** block can follow the catch block. This block of code gives the developer a chance to always run a bit of code regardless of whether an exception was thrown in the try block. Finally blocks are often used for cleanup of database connections and other resources.

For example, the following method attempts to open a file called “myFile.dat” and then attempts to read two integers from the file. If the file does not exist, a **FileNotFoundException**[1] is thrown by the File object. If the program attempts to read beyond the end of the file a **NoSuchElementException**[2] is thrown by the Scanner object.

[1] **FileNotFoundException** is an IOException and Java requires that you handle all IOExceptions with a try/catch block or by throwing the exception.

[2] **NoSuchElementException** is a RunTime exception and Java does not require that you handle RunTime exceptions. However, you should be aware of this exception and what causes it.

```
public void input() {
    try
    {
        Scanner reader = new Scanner(new File("myFile.dat"));      //Possible FileNotFoundException
        a = reader.nextInt();                    // where a is an instance field defined as an int
        b = reader.nextInt();                    // where b is an instance field defined as an int
    }
    catch (FileNotFoundException e)
    {
        System.out.println("Error opening data file!");        // Output an error message
        System.exit(0);                                        // Terminate the program
    }
}
```

## Changing The Delimiter

By default the Scanner class uses white space as a delimiter. White space consists of spaces, tabs, and new line (Enter) characters. To change the delimiter call the useDelimiter method of the Scanner class passing a string that contains the new delimiter to be applied.

```
Scanner reader = new Scanner(System.in);
reader.useDelimiter("/");
```

The two statements can easily be combined into a single statement:

```
Scanner reader = new Scanner(System.in).useDelimiter("/"));
```

Sometimes it is necessary to delimit on *more than one character*. For example, if you were reading from the keyboard and wanted to delimit on commas or forward slashes it would also be necessary to include the [ENTER] key as a delimiter. When the [ENTER] key is pressed two characters generated – a character 10 (return – "\r") and a character 13 (line feed – "\n"). To delimit based on a forward slash OR the [ENTER] key use the following statements:

```
Scanner reader = new Scanner(System.in);
reader.useDelimiter("/|\r\n");
```

The '|' character is located on the same key as the backslash key '\'.

**LAB 04 - ASSIGNMENT**

## Lab 04A - 60 points

### OBJECTIVE

WAP that reads a persons name and age from a data file (**"lab04a.dat"**). Output the person's name and the year the person was born.

### FIELD SUMMARY

- **String name** – a person's name.
- **int age** – a person's age.
- **int year** – the year the person was born.

### METHOD SUMMARY

- **main** – instantiate an object of your class. Call input, process, and output.
- **input** – declare a `Scanner` object and read a persons name and the year they were born from a data file.
- **process** – calculate the year the person was born.
- **output** – display the results on the console screen.

### SAMPLE DATA FILE INPUT

**Sally B. Goode/17**

### SAMPLE OUTPUT

**Sally B. Goode you were born in 1996.**

## Lab 04B - 70 points

### OBJECTIVE

WAP that reads three words from a data file (**"lab04b.dat"**). Output every possible combination of the three words.

### FIELD SUMMARY

- **String first** – the first word.
- **String second** – the second word.
- **String third** – the third word.

### METHOD SUMMARY

- **main** – instantiate an object of your class. Call input and output.
- **input** – declare a `Scanner` object and read three words from a data file
- **output** – display the results on the console screen. Use the tab escape sequence to separate the values.

### SAMPLE DATA FILE INPUT

**RED,BLUE,GREEN**

### SAMPLE OUTPUT

| | | |
|---|---|---|
| **RED** | **BLUE** | **GREEN** |
| **RED** | **GREEN** | **BLUE** |
| **BLUE** | **RED** | **GREEN** |
| **BLUE** | **GREEN** | **RED** |
| **GREEN** | **RED** | **BLUE** |
| **GREEN** | **BLUE** | **RED** |

## Lab 04C - 80 points

### OBJECTIVE

WAP that reads four sentences from a data file (**"lab04c.dat"**). Output the four sentences in the opposite order in which they were read.

### FIELD SUMMARY

- **String line1** – the first line of text.
- **String line2** – the second line of text.
- **String line3** – the third line of text.
- **String line4** – the third line of text.

### METHOD SUMMARY

- **main** – instantiate an object of your class. Call input and output.
- **input -** declare a `Scanner` object and read four sentences from a data file.
- **output** – display the results on the console screen.

### SAMPLE DATA FILE INPUT

**For my unconquerable soul.**
**I thank whatever gods may be**
**Black as the pit from pole to pole**
**Out of the night that covers me**

### SAMPLE OUTPUT

**Out of the night that covers me**
**Black as the pit from pole to pole**
**I thank whatever gods may be**
**For my unconquerable soul.**

## Lab 04D - 90 points

### OBJECTIVE

WAC that reads two sets of integers from a data file (**"lab04d.dat"**). Display the results of performing the following math operations on the two sets of numbers: add them together, subtract the second from the first, multiply them together, and divide the first by the second (floating point division). Format all doubles to display one decimal place. Suppress trailing zeros.

**Note**: The input is delimited by *commas* and *new line* keystrokes. Read `Changing The Delimiter` at the end of the documentation..

### SAMPLE DATA FILE INPUT

**7, 3**
**14,9**

### SAMPLE OUTPUT

**7 + 3 = 10**
**7 - 3 = 4**
**7 * 3 = 21**
**7 / 3 = 2.3**

**14 + 9 = 23**
**14 - 9 = 5**
**14 * 9 = 126**
**14 / 9 = 1. 6**

## Lab 04E - 100 points

### OBJECTIVE

WAP that reads a real number from a data file (**"lab04e.dat"**). Output the whole part and the fractional part of the number.

### SAMPLE DATA FILE INPUT

**9.25**

### SAMPLE OUTPUT

**The whole part is 9.**
**The fractional part is .25.**