File Input & Output (File I/O)

- File I/O in Java can be accomplished by using one of many built-in Java classes. (import java.io.*)
  - Reading input from a file:
    - **BufferedReader inFile = new BufferedReader(new FileReader("input.dat"));**

      Function: Reads text as a stream of characters.
  - Writing output to a file:
    - **PrintWriter outFile = new PrintWriter(new FileWriter("output.dat"));**

      Function: Prints formatted representations of objects to text output stream.

File I/O Exceptions

- Doing any file operations requires the programmer to either
  - use a try/catch block around every opening of a file and every read from the file
  - or write a throws clause on every method in the call stack up to and including the main method.
  - Exception is the most high level exception, so that can be used for IOExceptions and FileNotFoundExceptions. Also, IOException is pretty broad.

String.split method

- The StringTokenizer class allows an application to break a string into tokens and used to be the favored way to parse a string.
- But, StringTokenizer has been deprecated in favor of using the split method of the String class.
- String Tokenizer still works for the sake of "backwards compatibity", but you should use the String.split method to stay current.

String.split(" ")

- The StringTokenizer class has been "deprecated", meaning that its use is no longer advised (although it still works for the sake of "backward compatibility".
- The split method of the String class allows you to get the same result as the StringTokenizer.
- The "\\s" says to split the string by whitespace.
- The array of Strings called result (shown below) will contain an array of Strings after this code is run, with result[0]="this", result[1]="is", result[2]="a", and result[3]="test".

```
String[] result = "this is a test".split("\\s");
for (int x=0; x<result.length; x++)
    System.out.println(result[x]);
```

```
import java.io.*;
public class TestReadWriteSplit {
   public static void main (String[] args) throws Exception{
    BufferedReader fileIn = new BufferedReader
                       (new FileReader("pal.txt"));
    PrintWriter outFile = new PrintWriter
                      (new FileWriter ("palindrome.txt"));
    String line;
    String[] token;
    while ((line = fileIn.readLine()) != null) {
        token = line.replaceAll("[^a-zA-Z ]","")
             .toLowerCase().split("\\s+");
         String lcString = "";
         for(int i = 0; i < token.length; i++) {
           lcString += token[i];
         }
         outFile.println(lcString);
     }
     fileIn.close();
     outFile.close();
   }
}
```

Reading Command-Line Arguments

- Command-line arguments are read through the main method's array of Strings parameter, usually called args (or whatever you call it).

  IMPORTANT: *You must have a non-empty file in the same directory as the TestReadWriteSplit.class file called "input.txt" when you run this program!!* Also, any file called "output.txt" in the current directory will be overwritten. All files should be closed when you are done with them.

Reading Command-Line Arguments

- In the version of the TestReadWriteSplit program on the next slide, args[0] = "input.txt" and args[1] = "output.txt" during execution of the program.
- You can run this file in the Interactions window of DrJava (after it compiles with no syntax errors) by typing:

```
java TestReadWriteSplit input.txt output.txt
```

- Before this will work, you need to create a non-empty file in the same directory as your Java program called "input.txt". If you have any file in the directory that is already called "output.txt", its contents will be overwritten.

Reading Command-Line Arguments

```
import java.io.*;
public class TestReadWriteSplit {
   public static void main (String[] args) throws Exception{
    BufferedReader fileIn = new BufferedReader
                       (new FileReader(args[0]));
    PrintWriter outFile = new PrintWriter
                      (new FileWriter (args[1]));
    String line;
    String[] token;
    while ((line = fileIn.readLine()) != null) {
        token = line.replaceAll("[^a-zA-Z ]","")
             .toLowerCase().split("\\s+");
         String lcString = "";
         for(int i = 0; i < token.length; i++) {
           lcString += token[i];
         }
         outFile.println(lcString);
     }
     fileIn.close();
     outFile.close();
  }
}
```