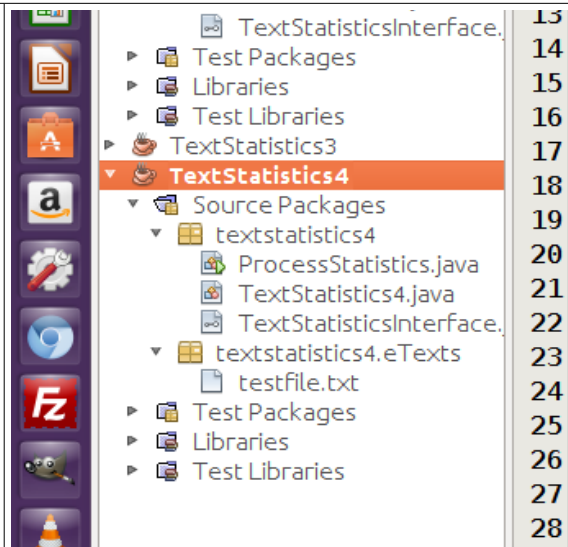


Project 3: Using the Tokenizer

Complete the Project 3: Getting Started.

Make sure you have the textstatistics project created.

Make sure you have successfully printed the number of lines in testfile.txt



The next portion of this project will require that we calculate the number of words in the document. We will use the StringTokenizer class to accomplish this.

Open the TextStatistics class and create a new private final String variable called DELIMITERS and assign the following string to this variable,

```
" ,,:\"&!?-_\\n\\t12345678910[{}]()@#$%&^*/+~"
```

```
public class TextStatistics4 implements TextStatisticsInterface{  
  
    private File textFile;  
    private Scanner fileScan;  
    private int lineCount;  
    private final String DELIMITERS = " ,,:\"&!?-_\\n\\t12345678910[{}]()@#$%&^*/+~";  
}
```

Declare a new StringTokenizer variable called tokenizer.

Declare a new int called wordCount

```
private File textFile;  
private Scanner fileScan;  
private int lineCount, wordCount;  
private final String DELIMITERS = " ,,:\"&!?-_\\n\\t12345678910[{}]()@#$%&^*/+~";  
private StringTokenizer tokenizer;
```

Click on the light bulb to the left of the StringTokenizer and import the required class

```
22 private int lineCount, wordCount;  
23 private final String DELIMITERS = " ,,:\"&!?-_\\n\\t12345678910[{}]()@#$%&^*/+~";  
24 private StringTokenizer tokenizer;  
25  
26 public TextStatistics4(File file){  
27
```

In the previous lesson you learned how to retrieve each line of the text document. The StringTokenizer will allow us to separate each line into tokens (or words). The tokenizer we declared above needs

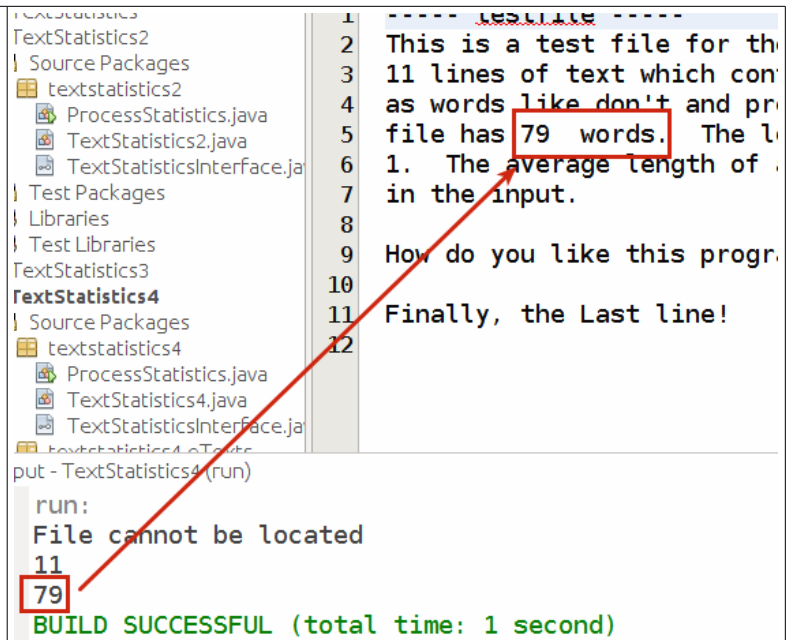
```
while(fileScan.hasNextLine()){  
    //fileScan.nextLine();  
    lineCount++;  
    tokenizer = new StringTokenizer(fileScan.nextLine(), DELIMITERS);  
}
```

Project 3: Using the Tokenizer

<p>two parameters, a line and a list of delimiters.</p> <p>Modify the while loop which counts the number of lines by initializing the tokenizer as follows,</p> <pre>tokenizer = new StringTokenizer(fileScan.nextLine(), DELIMITERS);</pre> <p>And deleting the “fileScan.nextLine()” statement.</p>	
<p>We will now count the tokens the same way we counted the lines.</p> <p>The loop below will iterate through each token on each line. Each time a token is found, wordCount will increment by one.</p> <pre>while(tokenizer.hasMoreTokens()) { tokenizer.nextToken(); wordCount++; }</pre>	<pre>while(tokenizer.hasMoreTokens()) { tokenizer.nextToken(); wordCount++; }</pre>
<p>Now that we have counted all the tokens, we can return the value in our method.</p> <p>Locate the getWordCount method. Replace the default code with the following,</p> <pre>return wordCount;</pre>	<pre>@Override public int getWordCount() { return wordCount; }</pre>
<p>Return to the ProcessStatistics class. To the if(file.exists()) clause, add the following,</p> <pre>System.out.println(ts.getWordCount());</pre>	<pre>if(file.exists()){ System.out.println(ts.getLineCount()); System.out.println(ts.getWordCount()); }</pre>

Project 3: Using the Tokenizer

Now, run your program. The number of tokens (words) will be printed. Open test file and confirm your results.



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes packages like `TextStatistics2`, `TextStatistics3`, and `TextStatistics4`. The code editor shows a test file with 12 lines of text. A red arrow points from the number 79 in the console output to the number 79 in the test file.

```
1  ----- TEST FILE -----
2  This is a test file for the
3  11 lines of text which contain
4  as words like don't and progra
5  file has 79 words. The last li
6  1. The average length of words
7  in the input.
8
9  How do you like this program?
10
11 Finally, the Last line!
12
```

run:
File cannot be located
11
79
BUILD SUCCESSFUL (total time: 1 second)