

Problem Set 2

Due: Saturday, September 21

Reading:

- Handout #6 (Enumerations)
- Contracts in the appendix of this problem set: `EnumTest`, `FileChars`, `FileLines`, `FileWords`, `StringChars`, `StringWords`.
- Contracts in the Sun Java 2 SKD, Version 1.4.0: `Character`, `Enumeration`, `String`, `StringBuffer`.

Overview: In this problem set, you will implement methods that determine various statistics for text files. Along the way, you will get experience with Java characters, strings, string buffers, arrays, and enumerations, as well as with writing, testing, and debugging classes written from scratch.

Each problem has a required part and a completely optional extra credit portion (which you may complete for extra credit points).

Download: You should download a copy of the directory `~cs230/download/TextStats` to begin this assignment. This directory contains implementations of the classes whose contracts are described in the appendix. In your local copy of this directory, you will create a class `TextStats` that contains your code for Problems 1, 2, and 3. For Problem 4, you will either create a class `MyStringWords` or `MyFileWords`.

Submission:

- For Problems 1, 2, and 3, your hardcopy should be your final version of `TextStats.java`.
- For Problem 4, your hardcopy should be your final version of either `MyStringWords.java` or `MyFileWords.java`.

Remember to include a signed cover sheet (found at the end of this problem set description) at the beginning of your hardcopy submission.

Your softcopy submission should be your entire `TextStats` directory.

Problem 1 [25]: Word Count

Background

Linux has a “word count” command, `wc`, that reports the number of lines, words, and characters in a file. For example, suppose that:

- `tricky.txt` is a file whose contents appears in Fig. 1. In the first line, `said` and `'Hello?'` are separated by a single tab character, as are `said` and `'Goodbye!'` in the second line.
- `initial.txt` is a file containing the initial segment of Dr.Seuss’s timeless classic *Green Eggs and Ham* shown in Fig. 2.
- `green.txt` is a file containing the full text of *Green Eggs and Ham*.

```
"He said      'Hello?',
but I said    'Goodbye!'",
she said.\\ \\ \\
```

Figure 1: The contents of `tricky.txt`.

```
I am Sam
I am Sam
Sam I am

That Sam-I-am!
That Sam-I-am!
I do not like
that Sam-I-am!

Do you like
green eggs and ham?

I do not like them,
Sam-I-am.
I do not like
green eggs and ham.
```

Figure 2: The contents of `initial.txt`.

Here is the result of invoking the `wc` command on these three files:¹

```
$ wc tricky.txt
 3   9  59 tricky.txt
$ wc initial.txt
16  40 185 initial.txt
$ wc green.txt
193 786 3465 green.txt
```

¹Assume that `$` is the Linux prompt, user input is in regular teletype font, and system output is in slanted font.

In each output line, the first number is the number of lines in the file, the second number is the number of words in the file, and the third number is the number of characters in the file.

The Linux documentation does not give a precise definition of “word”, but throughout this assignment we shall assume that a “word” is any contiguous sequence of alphanumeric characters (i.e., letters and digits), as well as certain “special characters” – namely, `'.'`, `'-'`, `'_'`, and `'\''` (i.e., the single quote character) – that occur between two alphanumeric characters. Any other characters, including special characters touching at least one non-alphanumeric character, are not part of any word. For example, the following sentences

```
Mary-Sue -- she wouldn't use "foo_bar", 'baz!quux', or @this*that%
to name 3.141 & $17.42. Would she?
```

contain the following 16 words:

```
Mary-Sue she wouldn't use foo_bar baz quux or this that
to name 3.141 17.42 Would she
```

Your Task

In this problem, your task is to implement a Java version of the Linux `wc` command. You should write a Java class method, `my_wc`, that takes a filename as an argument and displays the number of lines, words, and characters of the named file. Your method should be declared within the class `TextStats`, which you should create (in a file `TextStats.java`) as part of this problem.

```
$ java TextStats my_wc tricky.txt
tricky.txt has 3 lines, 9 words, and 59 chars.
$ java TextStats my_wc initial.txt
initial.txt has 16 lines, 40 words, and 185 chars.
$ java TextStats my_wc green.txt
green.txt has 193 lines, 786 words, and 3465 chars.
```

Notes:

- The results are formatted differently for `my_wc` than for `wc`, but the numbers are the same.
- For reading input from a file, you should use (one of) the enumerations described in the appendix.
- For full credit, your program should only read through the specified file exactly once. It is easier, but less efficient, to read through the file several times, collecting a different number on each pass. A two-pass or three-pass solution will be given partial credit.

You might want start with the multi-pass solution, and then transform it into a single-pass solution.

- Any Java file that uses classes implementing `Enumeration` (such as `TextStats`) must begin with the following import declaration:

```
import java.util.Enumeration;
```

If you fail to include the import declaration, the compiler will report that `Enumeration` is an unresolved symbol, as in the following compile-time error message:

```
TextStats.java:6: cannot resolve symbol
symbol   : class Enumeration
location: class TextStats
    Enumeration lines = new FileLines(filename);
    ^
```

- When extracting the next element of an enumeration via a call to `nextElement()`, recall that the compiler thinks that this method returns an `Object`. If you are attempting to use the element at some type other than `Object` (e.g., `String` or `Character`), you must downcast it. For example:

```
String s = (String) (enum.nextElement());
```

Extra Credit [10]

For extra credit, modify `my_wc` so that it prints its results using the same format as the Linux `wc` command (in which the three numbers are right justified in columns).

Problem 2 [25]: Character Frequency

In this problem, your task is to write a `TextStats` class method named `charFreq` that takes a filename as its single argument and displays the frequency (number of occurrences) for each character appearing at least once in the file. The characters and their frequencies should be displayed in ASCII order, one per line, in the format

```
'char' : freq
```

where *char* is a character representation and *freq* is the number of occurrences of the character in the file. The following *char* representations should be used for the special characters they denote: `\t`, `\n`, `\r`, `\'`, `\"`, and `\\`.

You should arrange that the `main` method of `TextStats`, when invoked with the two arguments `charFreq` and *filename*, should call your `charFreq` method on *filename*. For example, the result of `java TextStats charFreq tricky.txt` is shown in Fig. 3 and the result of `java TextStats charFreq initial.txt` is shown in Fig. 4.

Notes:

- You can maintain a histogram (i.e., occurrence count) for each character in a 128-element array indexed by the ASCII value of the character.
- You need to specially handle the display of the character representations `\t`, `\n`, `\r`, `\'`, `\"`, and `\\`.

Extra Credit [20]

For extra credit, modify `charFreq` so that it prints character/frequency lines sorted by frequency (from highest to lowest) rather than by ASCII value of the character. Characters with the same frequency should still be sorted by ASCII value of the character.

```
$ java TextStats charFreq tricky.txt
\t':2
\n':3
' ':4
'!':1
\"':2
\"':4
',':2
'':1
'/'::3
'?':1
'G':1
'H':2
'T':1
\"\":3
'a':3
'b':2
'd':4
'e':4
'h':1
'i':3
'l':2
'o':3
's':4
't':1
'u':1
'y':1
```

Figure 3: The result of the invocation `java TextStats charFreq tricky.txt`

```
$ java TextStats charFreq initial.txt
\n':16
' ':27
'!':3
',':1
'-'::8
'.'::2
'?':1
'D':1
'T':10
'S':7
'T':2
'a':21
'd':5
'e':11
'g':6
'h':6
'i':4
'k':4
'l':4
'm':17
'n':7
'o':8
'r':2
's':2
't':8
'u':1
'y':1
```

Figure 4: The result of the invocation `java TextStats charFreq initial.txt`

Problem 3 [25]: Word Frequency

In this problem, your task is to write a `TextStats` class method named `wordFreq` that takes a filename as its single argument and displays the frequency (number of occurrences) for the lowercase version of each word appearing in a file. The words and their frequencies should be displayed in alphabetical order, one per line, in the format

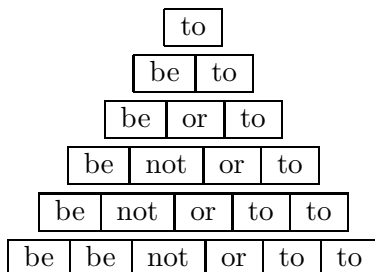
word:freq

where *word* is the word (consisting only of lowercase letters, digits, and special characters) and *freq* is the number of occurrences of each word the file.

You should arrange that the `main` method of `TextStats`, when invoked with the two arguments `wordFreq` and *filename*, should call your `wordFreq` method on *filename*. For example, the results of `java TextStats wordFreq` for `tricky.txt`, `initial.txt`, and `green.txt` are show in Figs. 6–7.

Notes:

- During the semester, we shall see many ways to solve this problem. For now, you should adopt the following strategy:²
 - Store (lowercase versions of) each word into a (growing) array of strings, sorted alphabetically. If the same word appears multiple times in the file, it should appear the same number of times in the array. For example, here are the sequences of arrays that should be generated in processing a file containing `To be or not to be`:



- To insert the next word in alphabetical order into the array, you should find the *insertion index* – the index at which the next word should be inserted. You can use either linear search or binary search to find this index.
 - Each insertion should create a brand new array. After processing *n* words, the current array should have exactly *n* elements.
 - Once you have an alphabetically sorted array of all words in the file, you can process the array to display the specified frequency table for `wordFreq`.
- To lowercase a string, you can use the method discussed in class, or simply invoke the appropriate method from the Java library.

²As we shall see later, this strategy is rather inefficient but there are much more efficient ways to solve the problem.

```
$ java TextStats wordFreq tricky.txt
but:1
goodbye:1
he:1
hello:1
i:1
said:3
she:1
```

Figure 5: The result of the invocation `java TextStats wordFreq tricky.txt`

```
$ java TextStats wordFreq initial.txt
am:3
and:2
do:4
eggs:2
green:2
ham:2
i:6
like:4
not:3
sam:3
sam-i-am:4
that:3
them:1
you:1
```

Figure 6: The result of the invocation `java TextStats wordFreq initial.txt`

Extra Credit [20]

For extra credit, modify `wordFreq` so that it prints word/frequency lines sorted by frequency (from highest to lowest) rather than by alphabetical order of the words. Words with the same frequency should still be sorted alphabetically.

```
$ java TextStats wordFreq green.txt
a:59
am:3
and:25
anywhere:8
are:2
be:4
boat:3
box:7
car:7
could:14
dark:7
do:37
eat:25
eggs:11
fox:7
goat:4
good:2
green:10
gren:1
ham:11
here:11
house:8
i:72
if:1
in:40
let:4
like:44
may:4
me:4
mouse:8
not:84
on:7
or:8
rain:4
sam:6
sam-i-am:13
say:5
see:4
so:5
thank:2
that:3
the:11
them:61
there:9
they:2
train:9
tree:6
try:4
will:21
with:19
would:26
you:34
```

Figure 7: The result of the invocation `java TextStats wordFreq green.txt`

Problem 4 [25]: Enumerations

In the previous problems, you have “worn the user hat” when using implementations of the `Enumeration` interface to process strings and files. In this problem, you will have a chance to “wear the implementer’s hat” by implementing (from scratch) a class implementing the `Enumeration` interface.

The `TextStats` directory you downloaded contains the compiled Java files `StringWords.class` and `FileWords.class` but it does *not* contain the source (`.java` files for the `StringWords` or `FileWords` classes). Your task in this problem is to implement *one* of these classes – you get to choose which one.

To avoid confusion, you should name your files `MyStringWords.java` and `MyFileWords.java` and name your classes `MyStringWords` and `MyFileWords`. This way, when you compile this files, you will not overwrite the existing files `StringWords.class` and `FileWords.class`.

Notes applicable to both `MyStringWords` and `MyFileWords`:

- In loops involving arrays and strings, it is often necessary to rely on the “short-circuit” nature of the boolean combinators `&&` and `||`. That is, in `exp1 && exp2`, if `exp1` evaluates to `false`, then `exp2` is never evaluated. Similarly, in `exp1 || exp2`, if `exp1` evaluates to `true`, then `exp2` is never evaluated. This is especially important in loop continuation conditions such as

```
((i < a.length) && (x > a[i])),
```

in which the `a[i]` in the second expression (if it were evaluated) would signal an array-out-of-bounds exception in the case where `(i < a.length)` is false.

- An important concept in implementing many objects, but especially enumerations, is an *invariant*: a relationship between the state variables that holds at the entry and exit of each method. See the specific notes below for appropriate invariants for `MyStringWords` and `MyFileWords`.

Notes on `MyStringWords`:

- The `main` method of `MyStringWords` should invoke `EnumTest.test` on an instance of `MyStringWords` class whose string is taken from argument to `main`. For an example, see Fig. 8. For other examples, experiment with `StringWords`, whose `main` method has the same desired behavior as that for `MyStringWords`.
- Handling the special characters `'.'`, `'-'`, `'_'`, and `'\''` is tricky. It’s a good idea to first handle purely alphanumeric examples before considering these.
- One way to implement `MyStringWords` is to use two instance variables: one to hold the string being processed, and the other to hold the “current index” into the string.
- The task of implementing `MyStringWords` is simplified if the following invariant is observed:

Let `s` be the string being processed and `i` be the current index. Then at the exit of the constructor method and at the entry and exit of every call to `hasMoreElements()` and `nextElement()`, one of the following two conditions should be true:

1. `i` should equal `s.length()` (indicating that there are no more elements); or

```

$ java MyStringWords "Mary-Sue -- she wouldn't use \"foo_bar\", 'baz!quux',\
> or @this*that% to name 3.141 & \$17.42. Would she?"a
Mary-Sue
she
wouldn't
use
foo_bar
baz
quux
or
this
that
to
name
3.141
17.42
Would
she
Total number of elements: 16

```

^aUnlike in Java, in Linux it is necessary to escape the exclamation point ! and the dollar sign \$ within a string to prevent the shell from interpreting them in a different way. Also, the backslash at the end of the first line is a way to continue a command to a second line. Each extra line of a multi-line command begins with the automatically generated prompt >.

Figure 8: An example invocation of the main method of MyStringWords

2. *i* should be the index of the first character (necessarily alphanumeric) in *s* of the next word to be enumerated.

Notes on MyFileWords:

- The main method of MyFileWords should invoke EnumTest.test on an instance of MyFileWords class whose string is taken from argument to main. For example, see Fig. 9. For other examples, experiment with FileWords, whose main method has the same desired behavior as that for MyFileWords.
- One way to implement MyFileWords is to use two instance variables: one that holds an instance of a FileLines enumeration that enumerates the lines of the file, and another that holds an instance of a StringWords or MyStringWords enumeration that enumerates the words from the “current” line. Care must be taken to handle the case where all words have been enumerated from the current line and the current line is followed by one or more empty lines (i.e., lines that contain no words).

Extra Credit [25]

Implement *both* MyStringWords and MyFileWords.

```
$ java MyFileWords tricky.txt
He
said
Hello
but
I
said
Goodbye
she
said
Total number of elements: 9
```

Figure 9: An example invocation of the main method of MyFileWords

A Documentation for CS230 Enumeration Classes

In CS230, we will be using several “home-brew” enumeration classes to manipulate files and strings. These classes are documented below.

A.1 EnumTest

The `EnumTest` class provides a single class method:

```
public static void test (Enumeration e);
```

Enumerates all the elements from `e` and displays the string representation of each element, one element per line. For a finite enumeration, after all elements have been enumerated, displays the number of elements that were enumerated.

For example, suppose that `StringChars` is a class enumerating all the characters from a string (see Sec. A.5). Then executing the statement

```
EnumTest.test(new StringChars("Hi, how are you?"));
```

yields the following output:

```
H
i
,
h
o
w

a
r
e

y
o
u
?
Total number of elements: 16
```

A.2 FileChars

The `FileChars` class is an implementation of the `Enumeration` interface that yields the characters of a given file one by one. Each enumerated character is wrapped in an instance of the `Character` class. In addition to the `hasMoreElements` and `nextElement` instance method required by implementations of `Enumeration`, `FileChars` supports the following constructor method and main testing method:

```
public FileChars (String filename);  
Create an enumeration that enumerates the characters of the file named by filename.
```

```
public static void main (String [] args);  
Invoke EnumTest.test on new FileChars(args[0]).
```

For example, suppose `test.txt` is a file with the following contents:

```
Why?  
Because!
```

Then here is an invocation of the `main` method of `FileLines` on `test.txt`:

```
$ java FileChars test.txt  
W  
h  
y  
?  
  
B  
e  
c  
a  
u  
s  
e  
!  
Total number of elements: 13
```

A.3 FileLines

The `FileLines` class is an implementation of the `Enumeration` interface that yields the lines of a given file one by one. Each line includes the terminating newline character, if present. In addition to the `hasMoreElements` and `nextElement` instance method required by implementations of `Enumeration`, `FileLines` supports the following constructor method and `main` testing method:

```
public FileLines (String filename);  
Create an enumeration that enumerates the lines of the file named by filename.  
  
public static void main (String [] args);  
Invoke EnumTest.test on new FileLines(args[0]).
```

For example, suppose `tricky.txt` is the file described in Problem 1. Then here is an invocation of the `main` method of `FileLines` on `tricky.txt`:

```
$ java FileLines tricky.txt  
"He said      'Hello?',  
  
but I said    'Goodbye!'",  
  
she said.  
  
Total number of elements: 3
```

In the above example, each pair of lines is separated by a blank line because each line includes a terminating newline in addition to the newline introduced for each line by `EnumTest.test`.

A.4 FileWords

The `FileWords` class is an implementation of the `Enumeration` interface that yields the words of a given file one by one. See Problem 1 for a definition of “word”. In addition to the `hasMoreElements` and `nextElement` instance method required by implementations of `Enumeration`, `FileWords` supports the following constructor method and `main` testing method:

```
public FileWords (String filename);  
Create an enumeration that enumerates the words of the file named by filename.  
  
public static void main (String [] args);  
Invoke EnumTest.test on new FileWords(args[0]).
```

For example, suppose `tricky.txt` is the file described in Problem 1. Then here is an invocation of the `main` method of `FileWords` on `tricky.txt`:

```
$ java FileWords tricky.txt  
He  
said  
Hello  
but  
I  
said  
Goodbye  
she  
said  
Total number of elements: 9
```

A.5 StringChars

The `StringChars` class is an implementation of the `Enumeration` interface that yields the characters of a given string one by one. Each character is wrapped in an instance of the `Character` class. In addition to the `hasMoreElements` and `nextElement` instance method required by implementations of `Enumeration`, `StringChars` supports the following constructor method and `main` testing method:

```
public StringChars (String s);  
Create an enumeration that enumerates the characters of s.  
  
public static void main (String [] args);  
Invoke EnumTest.test on new StringChars(args[0]).
```

For example:

```
$ java StringChars("Hi, how are you?")  
H  
i  
,  
  
h  
o  
w  
  
a  
r  
e  
  
y  
o  
u  
?  
Total number of elements: 16
```

A.6 StringWords

The `StringWords` class is an implementation of the `Enumeration` interface that yields the words of a given string one by one. See Problem 1 for a definition of “word”. In addition to the `hasMoreElements` and `nextElement` instance method required by implementations of `Enumeration`, `StringWords` supports the following constructor method and `main` testing method:

```
public StringWords (String s);  
Create an enumeration that enumerates the words of s.  
  
public static void main (String [] args);  
Invoke EnumTest.test on new StringWords(args[0]).
```

For example:

```
$ java StringWords "She asked, \"Did you pay $19.99 for 'Pocohantas'?\""  
She  
asked  
Did  
you  
pay  
19.99  
for  
Pocohantas  
Total number of elements: 8
```


Problem Set Header Page
Please make this the first page of your hardcopy submission.

CS230 Problem Set 2

Due Saturday September 21

Name:

Date & Time Submitted:

Collaborators (*anyone you worked with on the problem set*):

By signing below, I attest that I have followed the collaboration policy as specified in the Course Information handout.

Signature:

*In the **Time** column, please estimate the time you spend on the parts of this problem set. Please try to be as accurate as possible; this information will help me design future problem sets. I will fill out the **Score** column when grading your problem set.*

Part	Time	Score
General Reading		
Problem 1 [25]		
Problem 2 [25]		
Problem 3 [25]		
Problem 4 [25]		
Total		