CS2030 Practical Assessment #2 (Question)

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

CS2030 Practical Assessment #2

Problem Description

Before you proceed with the questions, please adhere to the following instructions:

- You need only design a single `Parser` class to be saved as `Parser.java`.
- In each level, `Parser.java` will be saved as `Parser1.java`, `Parser2.java`, etc. Only files numbered with a level will be uploaded to CodeCrunch.
- There **should not** be a `main` method in the `Parser` class. Any `main` method found will render the class uncompilable.
- A sample `jshell` session is provided for each level, as well as a `Test` driver class. Instructions for using the driver class will be given.
- You are to adhere strictly to each method signature. However, you may devise your own return types.

This task is divided into several levels. Read through all the levels to see how the different levels are related. **You need to complete ALL levels**.

Just remember to:

- check for output format correctness using the `diff` utility (see specific level for usage details). Note that only one test case is provided for this;
- save a copy of all source files into the appropriate level files (see specific level for usage details).

Level 1

Construct the `Parser` class and the method `parse(List lines)`, so as to output each line in `List lines`. For this and subsequent levels, you may assume that there is at least one line in `List lines`.

A sample `jshell` session is shown below:

```
jshell> /open Parser.java

jshell> List lines = Arrays.asList(new String[]{"one", "two three"})
lines ==> [one, two three]

jshell> Parser.parse(lines)
$.. ==> one
two three

jshell> Parser.parse(lines)
$.. ==> one
two three
```

From now on, your program will be tested using complete method chains starting with `Parser.parse(...)`, and `parse` will not be called again in the remaining part of the chain.

Click here to submit to CodeCrunch.

Check the format correctness of the output by typing the following Unix commands

```
$ javac Test1.java
$ java Test | diff - test1.out
```

Make a copy of your Java programs by typing the Unix command

```
$ cp Parser.java Parser1.java
```

Level 2

Define the methods `linecount()` and `wordcount()` so as to output the number of lines and words (consecutively sequence of non-space characters) respectively.

A sample `jshell` session is shown below:

```
jshell> /open Parser.java

jshell> List lines = Arrays.asList(new String[]{"one", "two three", ""})
lines ==> [one, two three, ]

jshell> Parser.parse(lines).linecount()
$.. ==> 3

jshell> Parser.parse(lines).wordcount()
$.. ==> 3

jshell> Parser.parse(lines).linecount().wordcount()
```

```
$.. ==> 1

jshell> Parser.parse(lines).wordcount().linecount()
$.. ==> 1
```

In the last method chain above, the final `linecount()` counts the number of lines of the output of `Parser.parse(lines).wordcount()`, which is `1`.

Hint: You may want to consider using the `split` method of the `String` class:

```
jshell> "one#two#three".split("#")
$.. ==> String[3] { "one", "two", "three" }

jshell> "one#two##three".split("#")
$.. ==> String[4] { "one", "two", "", "three" }
```

Click here to submit to CodeCrunch.

Check the format correctness of the output by typing the following Unix commands

```
$ javac Test2.java
$ java Test | diff - test2.out
```

Make a copy of your Java program by typing the Unix command

```
$ cp Parser.java Parser2.java
```

Level 3

Define a method `grab(String str)` that takes a string `str` and grabs only the strings with occurrences of `str`.

In addition, write a method `echo()` that simply echoes **all words separated by a single space on a single line**. Note that there is no trailing space at the end of each line.

A sample `jshell` session is shown below:

```
jshell> /open Parser.java

jshell> List lines = Arrays.asList(new String[]{"one", "two   three"})
lines ==> [one, two   three]

jshell> Parser.parse(lines).grab("e")
$.. ==> one
two  three

jshell> Parser.parse(lines).grab("ee")
$.. ==> two   three
```

```
jshell> Parser.parse(lines).grab("e").wordcount()
$.. ==> 3

jshell> Parser.parse(lines).grab("ee").wordcount()
$.. ==> 2

jshell> Parser.parse(lines).grab("z")
$.. ==>

jshell> Parser.parse(lines).grab("z").linecount()
$.. ==> 0

jshell> Parser.parse(lines).grab("z").wordcount()
$.. ==> 0

jshell> Parser.parse(lines).grab("z").linecount().grab("0").linecount()
$.. ==> 1

jshell> Parser.parse(lines).echo()
$.. ==> one two three

jshell> Parser.parse(lines).echo().grab("e t")
$.. ==> one two three

jshell> Parser.parse(lines).echo().grab("et")
$.. ==>
```

Note that `Parse.parse(lines).grab("z")` above does not output anything; hence the number of lines and number of words is zero.

Click here to submit to CodeCrunch.

Check the format correctness of the output by typing the following Unix commands

```
$ javac Test3.java
$ java Test | diff - test3.out
```

Make a copy of your Java program by typing the Unix command

```
$ cp Parser.java Parser3.java
```

Level 4

Define a method `chop(int start, int end)` that chops and retains each line from `start` to `end` inclusive of both. You may assume that `0 ≤ start ≤ end`.

As an example, invoking `chop(1, 4)` on `"aebecedef"` will result in `"aebe"`. Spaces are also retained if they are within the bounds. Note also that the starting position is `1` (not `0`).

A sample `jshell` session is shown below:

```
jshell> /open Parser.java

jshell> List lines = Arrays.asList(new String[]{"one", "two three"})
lines ==> [one, two three]

jshell> Parser.parse(lines).chop(2,2)
$.. ==> n
w

jshell> Parser.parse(lines).chop(2,5)
$.. ==> ne
wo t

jshell> Parser.parse(lines).grab("e").chop(0,10)
$.. ==> one
two three

jshell> Parser.parse(lines).grab("e").echo().chop(0,10)
$.. ==> one two th

jshell> Parser.parse(lines).chop(30,200)
$.. ==>

jshell> Parser.parse(lines).chop(30,200).linecount()
$.. ==> 2

jshell> Parser.parse(lines).chop(30,200).wordcount()
$.. ==> 0
```

Click here to submit to CodeCrunch.

Check the format correctness of the output by typing the following Unix commands

```
$ javac Test4.java
$ java Test | diff - test4.out
```

Make a copy of your Java program by typing the Unix command

```
$ cp Parser.java Parser4.java
```

Level 5

Define a method `shuffle` that shuffles each word in a line such that it obeys the following rules:

- The first occurring (and last occuring) letters (a to z; A to Z) are retained;
- All characters between the second letter to the second-last letter are left shifted one position with rotation.

In this way, the string `three` becomes `trehe` as shown in the following:

```
t  h <- r <- e  e
   v         ^
   |         |
   +---->----+
```

Here are some further examples:

consider the case where after the first letter there is an non letter

- the string `aren't` becomes `aenr't` (left shift with rotation on `"ren"`)
- the string `you'll` becomes `yu'lol` (left shift with rotation on `"ou'l"`)
- the string `T'was` becomes `T'aws` (left shift with rotation on `"wa"`)

The line `"two three"` consisting of two words becomes `"two trehe"` as shuffling is performed on the two words separately.

A sample `jshell` session is shown below:

```
jshell> /open Parser.java

jshell> List lines = Arrays.asList(new String[]{"one", "two three"})
lines ==> [one, two three]

jshell> Parser.parse(lines).shuffle()
$.. ==> one
two trehe
```

Define a `Test` class (say, in `Test5.java`) with a `main` method to read the following:

```
According   to   a   research   at   Cambridge   University,   it   doesn't   matter   in
what order the letters in a word are, the only important thing is that the first
and last letter be at the right place. The rest can be a total mess and you  can
still read it without problem.      This is because the human mind does not read
every     letter     by     itself     but     the     word     as     a     whole.
```

The output from shuffling is as follows:

```
Acordincg to a rsearceh at Cmbridgae Uiversitny, it desno't mttear in
waht oderr the ltteres in a wrod are, the olny iportanmt tinhg is taht the frsit
and lsat ltteer be at the rghit pacle. The rset can be a ttaol mses and you can
siltl raed it wthouit poblerm. Tihs is bcausee the hmaun mnid deos not raed
eervy ltteer by iseltf but the wrod as a wolhe.
```

Click here to submit to CodeCrunch.

Check the format correctness of the output by typing the following Unix commands

```
$ javac Test5.java
$ java Test < test5.in | diff - test5.out
```

Make a copy of your Java program by typing the Unix command

```
$ cp Parser.java Parser5.java
```

Level 6

Define an overloaded method `chop` that takes in a `String` followed by a sequence of positions of type `int`. For example, invoking `chop("e", 1, 2, 4)` will chop each line delimited by `"e"` and retain only the first, second and fourth portions. All portions retained (if present) are then output separated by the same delimiter. As an example, invoking `chop("e", 1, 2, 4)` on `"aebecedef"` will result in `"aebed"`. You may assume that there is at least one position, with all positions greater than zero, but not necessarily in order.

As a further example on this version of `chop`, below shows the string `"two three"` divided into parts enclosed within `[..]`. There are altogether three parts

```
[two thr]e[]e[]
```

with the second and third parts being empty strings. Furthermore, there is no fourth part. So applying `chop("e", 2, 3)` on `"two three"`, will output "e" (since the parts are empty), while applying `chop("e", 2, 4)` outputs an empty string as only the second part (empty string) is output, and there is no fourth part.

For the case where the delimiter is not found in the string, the entire string is output.

A sample `jshell` session is shown below:

```
jshell> /open Parser.java

jshell> List lines = Arrays.asList(new String[]{"one", "two three"})
lines ==> [one, two three]

jshell> Parser.parse(lines).chop("e", 1)
$.. ==> on
two thr

jshell> Parser.parse(lines).chop("e", 1, 2)
$.. ==> one
two thre

jshell> Parser.parse(lines).chop("e", 1, 2, 3)
$.. ==> one
two three

jshell> Parser.parse(lines).chop("e", 1, 2, 3, 4)
$.. ==> one
two three

jshell> Parser.parse(lines).chop("e", 2, 3)
$.. ==>
e

jshell> Parser.parse(lines).chop("e", 2, 4)
$.. ==>
```

```
jshell> Parser.parse(lines).chop("e", 2, 1, 3)
$.. ==> one
two three

jshell> Parser.parse(lines).chop("on", 1)
$.. ==>
two three

jshell> Parser.parse(lines).chop("on", 2)
$.. ==> e
two three

jshell> Parser.parse(lines).chop("hr", 2)
$.. ==> one
ee

jshell> Parser.parse(lines).shuffle().chop("eh", 1)
$.. ==> one
two tr
```

Click here to submit to CodeCrunch.

Check the format correctness of the output by typing the following Unix commands

```
$ javac Test6.java
$ java Test | diff - test6.out
```

Make a copy of your Java program by typing the Unix command

```
$ cp Parser.java Parser6.java
```