

CSE 11

discussion 6



HW6

1. Read a big file full of words
2. Ignore “common” words supplied in common.txt
3. Count the frequency of each word
4. Print the top N words

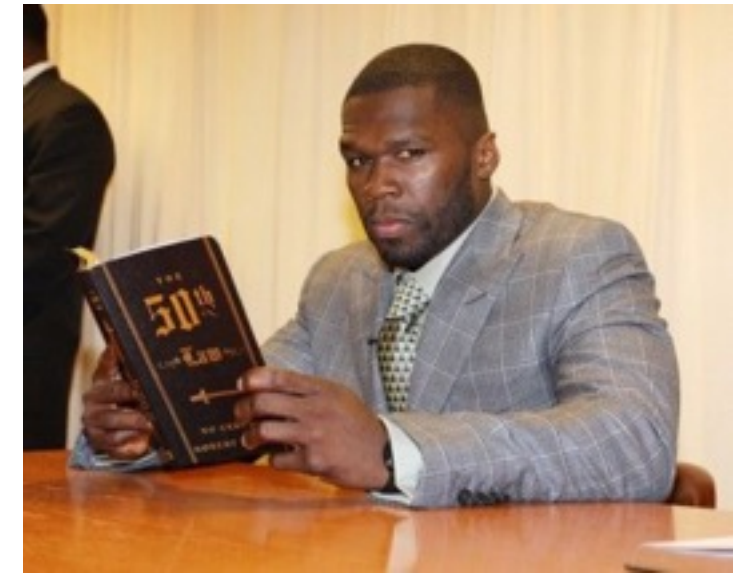
Command line arguments

- Now you finally get to use `String[] args`

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println(Arrays.toString(args));  
    }  
}
```

- When you run it:
\$ `java Main look at all these arguments`
`[look, at, all, these, arguments]`
- `args` contains everything after “`java <your class>`” on the command line
- `args` is a `String` array, so be sure to convert strings to numbers when necessary:
`Integer.parseInt(args[1])`

Reading stuff



- Your friend Scanner is back
`Scanner scnr = new Scanner(new File(input_file));`
- One problem: Scanner won't cut out punctuation

input: "punctuation; will! mess, you: up"

output:

`scnr.next()` -> "punctuation;"

`scnr.next()` -> "will!"

`scnr.next()` -> "mess,"

...

Deleting punctuation

- Everyone seems to be recommending *regular expressions*
- To process one string at a time, say, from `scnr.next()`:
(from Wednesday slides)

```
String str = "ToDAY!/?/@$";
```

```
String str2 = str.replaceAll("[^a-zA-Z]", "").toLowerCase();
```



regular expression (regex)

- Even better...just tell Scanner to skip all punctuation and spaces:
`scnr.setDelimiter(...);`
(I'll let you figure out the regex)
(Hint: it should detect non-word characters repeating 1 or more times)
 - `setDelimiter(regex)` = "each word is separated by *regex*" (default: whitespace)
 - Now whenever you call `scnr.next()` it removes the punctuation!

Ignoring common words

- Read in common.txt the same way you would with the main word file
- Put each word into a *HashSet*
- HashSet: you need some way to check whether a value is present in a collection, but you don't care what it represents

```
HashSet<String> set = new HashSet<>();  
set.add("water");  
set.add("beer");  
set.add("vodka");  
set.add("everclear");  
set.add("heroin");
```

```
//got any beer?  
set.contains("beer"); //→ true
```

- What if you used ArrayList instead of HashSet?

```
ArrayList<String> set_array = new ArrayList<>();  
...  
set_array.contains("beer");
```
- ArrayList contains() method: **O(n)**
HashSet contains() method: **O(1)** ← much faster

Counting each word with HashMaps

- You've got each word, you know how to skip common words
- Now you need to keep a count for each word
- *HashMaps: Arrays on steroids*
- Array and List: you can only look up something by its position in the list.

```
List<Integer> list = new ArrayList<>();  
list.add(0, 1337);  
list.add(1, 69);  
list.add(2, 420);
```

```
list.get(2); //--> 420
```

- HashMaps: index can be many things, like String:

```
Map<String, Integer> a = new HashMap<>();  
a.put("leet", 1337);  
a.put("...", 69);  
a.put("weed", 420);
```

```
a.get("weed"); //--> 420
```

Using the HashMap's count

- You've got a HashMap with the number of occurrences of each word
- How do you get the top N words?
- The easy/slow way:

```
for(int i=0;i<N;i++){  
    for (String word : word_count.keySet()) {  
        //keep track of the maximum word_count.get(word)  
        //find the string with the highest count  
    }  
    //delete the maximum  
}
```

this is known as a “foreach” loop
think of it as “for each word in word_count.keySet()”



- Just delete the maximum N times. But this is $O(n^2)$

Questions

The Regex (regular expression)

- You want to find a string, you sort of know what it looks like, but not exactly

"CSE11"

"CSE12"

"CSE20"

"CSE21"

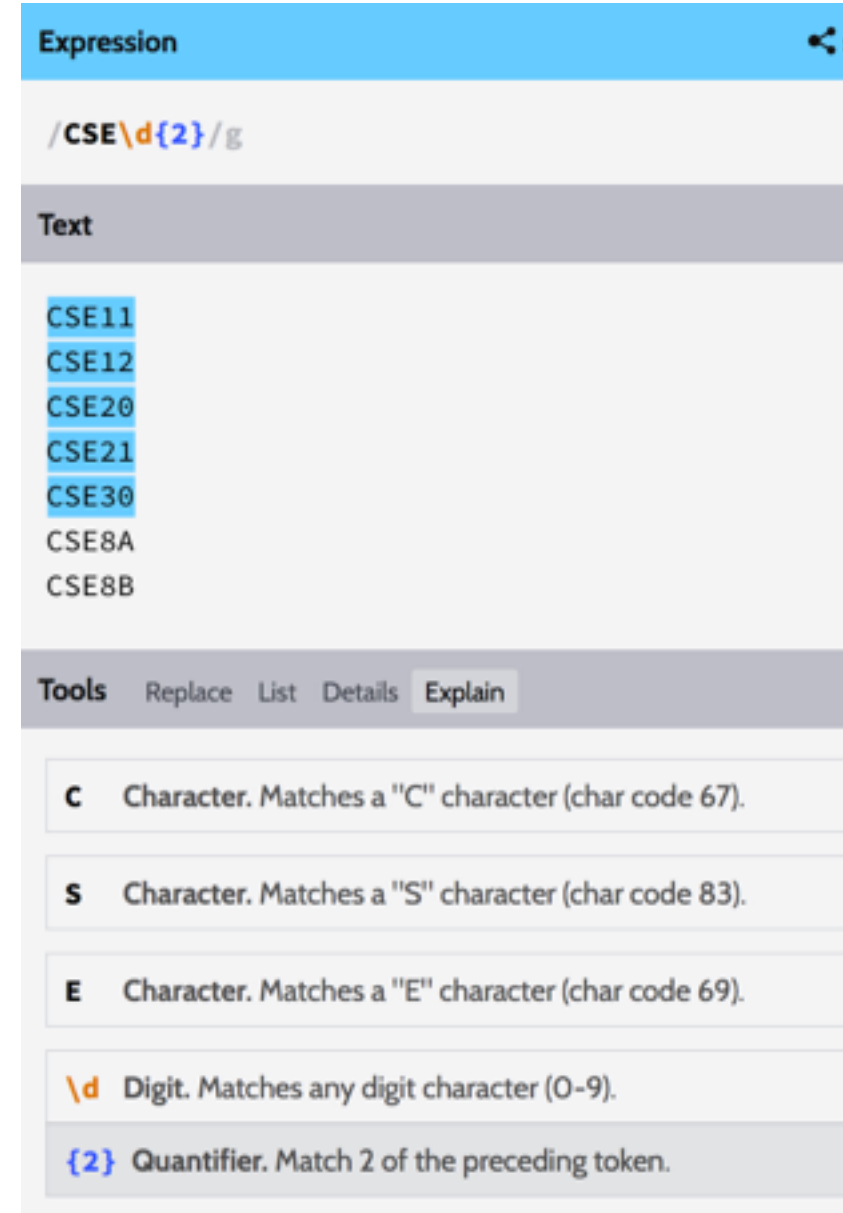
"CSE30"

...

- I want the letters "CSE" followed by 2 numbers
- It's just: `CSE\d{2}`
 - `CSE` just the letters "CSE"
 - `\d` a digit 0-9
 - `{2}` repeat the last character 2 times
- (demonstration: <http://regexpr.com/>)

Try it yourself

- <http://regexpr.com/>
(demonstration)
- Do not attempt to memorize regexes. You must learn by doing.
- (as with most things in computer science...)



The screenshot shows the regexpr.com website interface. At the top, the 'Expression' field contains the regex `/CSE\d{2}/g`. Below it, the 'Text' field contains a list of identifiers: CSE11, CSE12, CSE20, CSE21, CSE30, CSE8A, and CSE8B. The first five are highlighted in blue, indicating they match the regex. At the bottom, the 'Tools' section is active, showing an 'Explain' tab with a breakdown of the regex components: 'C' (Character, Matches a "C" character (char code 67)), 'S' (Character, Matches a "S" character (char code 83)), 'E' (Character, Matches a "E" character (char code 69)), '\d' (Digit, Matches any digit character (0-9)), and '{2}' (Quantifier, Match 2 of the preceding token).

Expression
<code>/CSE\d{2}/g</code>

Text
CSE11
CSE12
CSE20
CSE21
CSE30
CSE8A
CSE8B

Tools	Replace	List	Details	Explain
C	Character. Matches a "C" character (char code 67).			
S	Character. Matches a "S" character (char code 83).			
E	Character. Matches a "E" character (char code 69).			
\d	Digit. Matches any digit character (0-9).			
{2}	Quantifier. Match 2 of the preceding token.			

Regex in Java

- Use a double backslash “\\” in a Java string to represent 1 backslash

```
String s = "separated393by0numbers";  
String[] split = s.split("\\d+");  
System.out.println(Arrays.asList(split));
```

output:
[separated, by, numbers]

- Also, the split() method is incredibly useful

Mostly irrelevant but hilarious exchange on Stack Overflow about regexes

- <http://stackoverflow.com/questions/1732348/regex-match-open-tags-except-xhtml-self-contained-tags>

(the top answer)

Sorting HashMap contents (optional)

- The faster way to print the top N words is to sort the contents of the HashMap first
- First, use HashMap's *entrySet()* and get a list of each word and its associated count:

```
ArrayList<Entry<String, Integer>> words_array =  
    new ArrayList<>(words.entrySet());
```

- Ok, but how do you sort a list of Entry<String, Integer>?
- The secret is to use this second argument of **Collections.sort**:

```
sort(List<T> list, Comparator<? super T> c)  
Sorts the specified list according to the order induced by the  
specified comparator.
```

- You can tell sort how to compare any two elements while sorting

Sorting HashMap

- When you sort something you need to know how to compare two things
sort {2, 1} —> *is 2 less than 1? No, move it after 1*
- `ArrayList<Entry<String, Integer>>` is basically an array with 2 values each:
`{"shall", 10781}, {"unto", 9033 }, {"lord", 8018 }`
- Sort doesn't know how to compare values of this kind
- You need to make your own *Comparator* so that sort knows how to rank `Entry<String, Integer>` by the count, not the word.

```
Collections.sort(words_array,  
    new Comparator<Entry<String, Integer>>() {  
        public int compare(Entry<String, Integer> o1, Entry<String, Integer> o2) {  
            //return -1 if o1 < o2  
            //return 0 if equal  
            //return 1 if o1 > o2  
        }  
    }  
);
```