**CMSC 214**

**Project #6**

**Derya O. Kurin**

1. **CountingJavaKeywords (Covers Chapter 21)**

**UML Diagram:**

Graphical user interface, application

Description automatically generated

**Pseudocode:**

Main method(args: String[]):void

Create an instance of Scanner class as input

Ask user to enter a file name

Create a file variable, tore the nextLine in it

IF file exists AND file extension equals to .java

Output the total Java keywords in the file

ELSE

Output: File name does not exist

ENDIF

END Main

Method countKeywords(file:File):int

Declare an array if Strings of all Java keywords

Create a set of Strings keywordSet and assign it to a new HashSet of keywordString List.

Declare and assign count to 0.

Set the Scanner object for the file input

WHILE the input file has a next item,

Store the next word in a String variable word

If word equals //

Pass to nextLine

ELSE IF word contains \

WHILE input file has a next word AND next input item does not contain \

ENDWHILE

ELSEIF word contains /\*

WHILE there is next word AND next input item does not contain \*/

ENDWHILE

ELSEIF keywordSet contains any word

Increment count by 1

ENDIF

ENDWHILE

Return count

END countKeywords method

**Lessons Learned:**

I learned how to use HashSet in my code. I learned that HashSets have a load factor and by default they are 0.75. when we reach towards the end of its capacity while adding objects in it, it doubles its size. And another thing I learned about HashSets are we can change this default value to anything between 0.0 and 1.0 depending on our program and our objectives with it. Once we created a hash set consisting of the Java keywords it was easy to check whether the file input word was in the keywordSet using contains method. We preferred using a hashset since each java keyword in the list was unique(not double values).

In an addition to the example provided in the book, I added the code necessary for not to count the java keywords in the comments.

For this I first counted the keywords in that java file and then added a few keywords in // comments and run the program again and got the same result. I then added keywords between /\* and \*/ comment symbols and again got the same result indicting that my code was actually working.

**Test Plan:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Input | Actual Input | Output | Actual Output | Did it pass the test? |
| 1 | Enter a Java source file: java\_source\_code1.java | Enter a Java source file: java\_source\_code1.java | 6 | 6 | Yes |
|  | After adding java keywords as a comment  Both using // and /\* \*/ | Enter a Java source file: java\_source\_code1.java | 6 | 6 | Yes |
| 2 | Enter a Java source file: java\_source\_code2.java | Enter a Java source file: java\_source\_code2.java | 13 | 13 | Yes |
|  | After adding java keywords as a comment  Both using // and /\* \*/ | Enter a Java source file: java\_source\_code2.java | 13 | 13 | Yes |

**Screen Shots:**

Text

Description automatically generated

1. **Execution time for sorting**

**UML Diagram:**

**Diagram

Description automatically generated**

**Lessons Learned:**

In this project, I had the chance to learn about all the common sorting algorithms, how to implement them and how much time they generally take to sort large size arrays.

Implementing their algorithm and being able to see how they actually perform was a very useful experience. I think it was much more helpful for me to visually see their run time than reading about their time complexity in the books.

It gave me an overall picture of what sorting algorithms are more preferable depending on the case.

**Screen Shots:**

**A screen shot of a computer

Description automatically generated**

**Check List:**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** |  | **Y/N** | **Comments** |
|  | **Source java files** |  |  |
|  | **Compressed files:** |  |  |
|  | FirstInitialLastName\_Project6\_1\_Moss.zip | **Y** |  |
|  | FirstInitialLastName\_Project6\_2\_Moss.zip | **Y** |  |
|  | FirstInitialLastName\_Project6\_doc.zip | **Y** |  |
|  | **Program compiles** | **Y** |  |
|  | **Program runs** | **Y** |  |
|  | **Checklist is completed and included in the Documentation** | **Y** |  |
|  | **Documentation file:** |  |  |
|  | **Comprehensive Test Plan** | **Y** |  |
|  | **Screenshots based on Test Plan** | **Y** |  |
|  | **UML Diagram** | **Y** |  |
|  | **Algorithms/Pseudocode** | **Y** |  |
|  | **Flowchart** | **N/A** |  |
|  | **Lessons Learned** | **Y** |  |