**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 execution time than reading about their time complexity in the books.

It gave me an overall picture of what sorting algorithms are more preferable for at least int values and this might change depending on the data type of the array we are sorting.

I learned that the most convenient sorting algorithms are Quick Sort, Merge Sort, Radix Sort and Heap sort in this order compared to Selection Sort and Bubble Sort.

Radix Sort is a linear sorting algorithm

**Pseudocode for sorting algorithms:**

**Procedure: selectionSort(list: int[]):void**

Find the smallest integer. Swap it with the first integer.

Find the second-smallest integer. Swap it with the second integer.

Find the third-smallest integer. Swap it with the third integer.

Repeat finding the next-smallest integer, and swapping it into the correct position until the array is sorted.

ENDPROCEDURE

**Procedure: radixSort(list: int[], maxOrder: int):void**

FOR order 1 to maxOrder increment order by its multiples with 10

Declare an ArrayList of Integers as bucket and set its capacity to 10

FOR i 0 to bucket length

Add the ith list item to bucket’s item: remaining of 10 from the division of ith item by order

ENDFOR

Declare and assign k to 0

FOR I from 0 to bucket length

IF ith bucket item is null

FOR j is from 0 to bucket’s ith item’s size

Assign list’s iterated k’th item to bucket’s ith element’s jth item

ENDFOR

ENDIF

ENDFOR

ENDFOR

ENDPROCEDURE

**Procedure: bubbleSort(list: int[]): void**

Declare and assign needNextPass to true

FOR k from 1 to list.length AND needNextPass is true

Assign needNextPass to false

FOR I from 0 to lsit length – k, increment I by 1

IF ith list item is greater than i+1 st list item

Swap ith list item with (i+1)th list item

Set needNextPass to true

ENDIF

ENDFOR

ENDFOR

ENDPROCEDURE

**Procedure: mergeSort(list: int[]): void**

IF list length is greater than 1

Create an int array for the first half of list called firstHalf

Call arraycopy with list, 0, first half, 0, half size of the list

Call mergeSort with the firstHalf

Declare an int secondHalfLength and assign it to

Subtraction of half-length from the list length

Create a new int array secondHalf with a capacity of secondHalfLength

Call arraycopy with list, half length of list, second half, 0, secondHalfLength

Call mergeSort with secondHalf

Call merge with fistHalf, secondHalf, and list

ENDIF

ENDPROCEDURE

**Procedure: merge(list1: int[], list2:int[], temp: int[]):void**

Declare current1 and assign it to 0 (for index iteration for list1 > will stand for current index of list1)

Declare current2 and assign it to 0 (for index iteration for list2 > will stand for current index of list2)

Declare current3 and assign it to 0 (for index iteration for list3 > will stand for current index of list3)

WHILE current list1 element is less than list1 length AND current list2 element is less than list2 length

IF current element of list1 is less than current list2 element

Store the next list1 item in temp’s next index.

ELSE

Store the next list2 item in temp’s next index

ENDIF

ENDWHILE

WHILE current 1 is less than list1 length

Store next item of list1 in next temp item

ENDWHILE

WHILE curren2 is less than list2 length

Store the next list2 element in next temp index

ENDWHILE

ENDPROCEDURE

**Procedure quickSort(list: int[]):void**

Call quicksort with 3args with parameters: list, 0 and list’s last index

ENDPROCEDURE

**Procedure quickSort(list: int[], first: int, last: int): void**

IF last is greater than first

Declare an int pivotIndex and assign it to the return value of partition method called with params: list, first and last

Call quicksort with 3args for the first half of the list until the pivotIndex

Call quicksort with 3 args for the last half of the list after the pivotIndex

ENDIF

ENDPROCEDURE

**Procedure: partition(list: int[], first: int. last: int): int**

Set pivot to first item of the list

Set low to first +1

Set high to last

WHILE high is greater than low

WHILE low is less than or equal than high AND low item of list is less than or equal than pivot

Increment low

ENDWHILE

WHILE low is less than or equal than high AND high item of list is greater than pivot

Decrement high

ENDWHILE

IF high is greater than low

Swap high index of list with low index of list

ENDIF

WHILE high is greater than fisrt AND list’s high index elemnt is greater than or equal to pivot

Decrement high

ENDWHILE

IF pivot is greater than the high index of list

Set index first of list to high index of list

High index of list to pivot

Return high

ELSE

Return first

ENDIF

ENDPROCEDURE

**Procedure heapSort(list: int[]): void**

Create a heap of Integers called heap

FOR I index from 0 to list length

Add list items to heap

ENDFOR

FOR each index I from list’s last index to 0

Assing ith element of list to heap with one less element

ENDFOR

Declare a Heap class that extend Comparable

Create an ArrayList list of E elements

Define no arg constructor for Heap class

Define 1-arg constructor with an arg of array E objects

Add each object element to the heap

END constructor

Create procedure add(newObject: E): void

Append newObject to heap

Set currentIndex to the last index of heap

WHILE currentIndex is greater than 0

Set parentIndex half value of the currentIndex

IF the current object is greater than its parent

Swap currentIndex with parentIndex elemnt in list

ELSE

Stop the if execution

ENDIF

Set currentIndex to parentIndex

ENDWHILE

ENDPROCEDURE

**Procedure: remove():E**

If list is empty return null

Set removedObject to the first element of the list

Call set on list with 0 and the last item of the list

Remove the last item from the heap list

Set currentIndex to 0

WHILE currentIndex is less than the list size

Set leftChildIndex to double of the currentIndex incremented by 1

Set rightChildIndex to double of the currentIndex incremented by 2

Find the maximum between the two child

IF curentIndex element is less than maxIndex element in the heap

Swap the list’s maxIndex with currentIndex

ELSE

Stop execution of IF statement

ENDIF

ENDWHILE

Return the removedObject

ENDPROCEDURE

**Procedure: getSize(): int**

Return the list’s size

ENDPROCEDURE

ENDCLASS

ENDPROCEDURE

**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** |  |