Project 1 (in Java): Giving a file contains English words, the task is to construct an ordered linked list (in ascending order) using insertion sort, then, to locate the middle node in the linked list.

** You are given two test data: LLMiddleNode_Data1and LLMiddleNode_Data2 to run your program.
What you have to do:
1) Implement your program according the specs below.
2) Debug and test your program using LLMiddleNode_Data1 until your program produces correct output
3) Run your program with LLMiddleNode_Data2
4) Include in your hard copy (pdf file)
- cover page
- source code
- outFile from LLMiddleNode_Data1
- debugFile from LLMiddleNode_Data1
- outFile from LLMiddleNode_Data2
- debugFile from LLMiddleNode_Data2

****

Language: Java

****

Project Name: Linked List & Middle Node
Project points: 10 pts
Due Date:
-0 (10/10pts): on time, 2/8/2023 Wednesday before midnight
-1 (9/10 pts): 1 day late: 2/9/2023 Thursday before midnight
-2 (8/10 pts): 2 days late: 2/10/2023 Friday before midnight
-10/10 pts: non-submission: 2/10/2023, Friday after midnight

*** Name your soft copy and hard copy files using the naming convention (according to the Project Submission Requirements).
*** All on-line submission MUST include Soft copy (*.zip) and hard copy (*.pdf) in **the same email attachments** with correct email subject as stated in the project submission requirement; otherwise, your will be rejected. submission

****

II. inFile (use args [0]): A text file contains English words in various format. (You may NOT modify the file format to suit your program!)
Outputs:
a) outFile1 (use args[1]): your program outputs in a text file, includes :
i) The completed sorted linked list, five words in one text line;
ii) The word in the middle node with caption.
b) deBugFile(use args[2]): All debugging outputs, so you may get partial credits in case your program does not work completely.

****

III. Data structure:

****

- a listNode class
- (string) data
- (listNode) next
Methods:
- constructor (data): create a new listNode with (data, null)
- a LList class
- (listNode) listHead // points to the dummy node.
- (listNode) middleNode // points to the middle node in the list.
methods
- constructLL (…) // See algorithm below.
- listInsert (…) // See algorithm below.
- findSpot (…) // Use the findSpot algorithm steps taught in class.

- printList (listHead, File) // File can be outFile or deBugFile
    // print the list to File, from listHead to the end of the list in the following format:
    listHead → ("dummy", next's data) → (this node' data, next's data) →… → NULL

    For example:
    listHead → (dummy, Anne) → (Anne, Bobby) → (Bobby, Dean). →.... → NULL

- findMiddleNode (…) // see algorithm below.

****

IV. Main(…)

****

Step 1: inFile ←open with args[0]
    outFile ← open with args[1]
    deBugFile ← open with args[2]
Step 2: listHead ← get a new listNode with ("dummy"), as the dummy node for listHead to point to.
Step 3: constructLL (listHead, inFile, deBugFile)
Step 4: printList (listHead, outFile) // Print the complete list to outFile
Step 5: middleNode ← findMiddleNode (listHead, deBugFile)
Step 6: if middleNode != null // in case the list is empty
        outFile ← middleNode's data // with caption "the word in the middle of list is"
Step 7: Close all files

****

V. constructLL (listHead, inFile, deBugFile)

****

Step 0: deBugFile ← output "In constructLL method" to deBugFile // debug prints
Step 1: data ← read one word from inFile
Step 2: newNode ← get a new listNode (data)
Step 3: listInsert (listHead, newNode)
Step 4: printList (listHead, deBugFile) // debug prints linked list after every insertion.
Step 5: repeat step 1 – step 4 until the end of inFile

****

V. listInsert (listHead, newNode, deBugFile)

****

Step 0: deBugFile ← output "In listInsert method" // debug prints
Step 1: Spot ← findSpot (listHead, newNode)
    deBugFile ← output "Returns from findSpot where Spot.data is"
Step 2: newNode's next ←Spot's next
    Spot's next ← newNode

****

VII. findMiddleNode (listHead, deBugFile)

****

Step 0: deBugFile ← output "In findMiddleNode method" to deBugFile // debug prints
Step 1: walker1 ← listHead's next
    walker2 ← listHead's next
Step 2: if walker2 != null *and* walk2's next != null
        walker1 ← walke1's next
        walker2 ← walker2's next's next // walker2 walks twice as fast as walker1
Step 3: deBugFile ← output "walker1's data is" … to deBugFile.
Step 4: repeat step 2 to step 3 until condition failed
Step 5: return walker1