PSEUDOCODE FOR TEXT PARSER CLASS

```
CLASS TextParser
  PRIVATE LinkedList ll

  FUNCTION readTextFile(filePath: String) -> String
    BEGIN
      fileContent = ""
      TRY
        Open the file at 'filePath'
        WHILE not end of file
          Read a line from the file
          Append the line to 'fileContent' with a space separator
        END WHILE
        Close the file
      CATCH IOException
        Print the error stack trace
      END TRY
      RETURN fileContent
    END FUNCTION

  FUNCTION readAndAddToLL()
    BEGIN
      filePath = "/Users/egj/Desktop/TextParser2/pg174.txt"
      text = readTextFile(filePath)
      words = split 'text' by space character
      FOR EACH word IN words
        Insert 'word' into the linked list 'll'
      END FOR
    END FUNCTION

  FUNCTION getSize() -> Integer
    BEGIN
      RETURN ll's size
    END FUNCTION

  FUNCTION getWordCount()
    BEGIN
      targetWords = ["portrait", "persian", "dorian", "experimental", "magnetic"]
      FOR EACH word IN targetWords
        wordFound = false
        current = ll's head
        WHILE current is not null
          IF current's word is equal to 'word'
            Print 'word' and current's instance count
            Set 'wordFound' to true
            BREAK the loop
          END IF
          Move 'current' to the next node
        END WHILE
        IF 'wordFound' is false
          Print 'word' and "Not found"
        END IF
```

```
    END FOR
  END FUNCTION

  FUNCTION getWordsMoreThan20()
    BEGIN
      current = ll's head
      WHILE current is not null
        IF current's instance count is greater than 20
          Print current's word and instance count
        END IF
        Move 'current' to the next node
      END WHILE
    END FUNCTION

  FUNCTION getMostFrequentWord()
    BEGIN
      maxCount = 0
      mostFrequentWord = ""
      current = ll's head
      WHILE current is not null
        IF current's instance count is greater than 'maxCount'
          Set 'maxCount' to current's instance count
          Set 'mostFrequentWord' to current's word
        END IF
        Move 'current' to the next node
      END WHILE
      IF 'mostFrequentWord' is not empty
        Print 'mostFrequentWord' and 'maxCount'
      END IF
    END FUNCTION

  FUNCTION main()
    BEGIN
      TP = create a new TextParser
      Call TP's readAndAddToLL function
      Call TP's getWordsMoreThan20 function
      Call TP's getSize function
      Call TP's getMostFrequentWord function
      Call TP's getWordCount function
      Print "PROCESSES COMPLETED"
    END FUNCTION
END CLASS
```

PSEUDOCODE FOR LINKED-LIST CLASS

```
CLASS LinkedList
  PRIVATE LinkedListNode head
  PRIVATE Integer size

  FUNCTION LinkedList()
    BEGIN
      // Constructor for the LinkedList class
```

```
    head = null
    size = 0
  END FUNCTION

  FUNCTION insert(word: String)
    BEGIN
    // Pre-process the 'word' before insertion:
    // Convert the word to lowercase and remove any non-alphanumeric characters (except
spaces).
    word = convertToLowercaseAndRemoveNonAlphanumericCharacters(word)
    IF head is null
      // If the linked list is empty, create the first node (head) with the provided 'word'.
      head = createNewNodeWithWord(word)
    ELSE
      // If the linked list is not empty, traverse through it to check if the 'word' already exists.
      current = head
      WHILE current's next is not null
        IF current's word is equal to 'word'
          // If the 'word' already exists in a node, increment its instance count and return.
          incrementInstanceCount(current)
          RETURN
        END IF
        // Move to the next node in the linked list.
        current = current's next
      END WHILE

      // If the 'word' doesn't exist in the linked list, add a new node to the end of the list.
      addNewNodeToEndOfList(current, word)
    END IF

    // Increment the size of the linked list to reflect the new insertion.
    incrementSize()
  END FUNCTION

  FUNCTION getSize() -> Integer
    BEGIN
    PRINT size
    RETURN size
  END FUNCTION

  FUNCTION getWordsMoreThan20()
    BEGIN
    PRINT "Words that occur more than 20 times in the list:"
    current = head
    size = 0
    WHILE current is not null
      IF current's instance count is greater than 20
        PRINT current's word and current's instance count
        INCREMENT size
      END IF
      current = current's next
    END WHILE
    PRINT "End of List - (Amount of words occurring more than 20 times (" + size + "))"
  END FUNCTION
```

```
  FUNCTION getLongestWord() -> String
    BEGIN
      current = head
      longestWord = ""
      WHILE current is not null
        IF length of current's word is greater than length of longestWord
          longestWord = current's word
        END IF
        current = current's next
      END WHILE
      PRINT "LONGEST WORD IN THE LIST: " + longestWord
      RETURN longestWord
    END FUNCTION

  FUNCTION getMostFrequentWord() -> String
    BEGIN
      current = head
      mostFrequentWord = ""
      mostFrequentWordCount = 0
      WHILE current is not null
        IF current's instance count is greater than mostFrequentWordCount
          mostFrequentWordCount = current's instance count
          mostFrequentWord = current's word
        END IF
        current = current's next
      END WHILE
      PRINT "MOST FREQUENT WORD IN THE LIST: " + mostFrequentWord
      RETURN mostFrequentWord
    END FUNCTION

  FUNCTION getWordCount()
    BEGIN
      PRINT "AMOUNT OF TIMES THE FOLLOWING WORDS OCCUR (ASSUMING THEY'RE
FOUND) - (PERSIAN, PORTRAIT, DORIAN, EXPERIMENTAL, AND MAGNETIC):"
      PRINT "WORD(S) THAT ARE NOT FOUND ARE NOT DISPLAYED"
      current = head
      WHILE current is not null
        IF current's word is "portrait" THEN
          PRINT "PORTRAIT: " + current's instance count
        ELSE IF current's word is "persian" THEN
          PRINT "PERSIAN: " + current's instance count
        ELSE IF current's word is "dorian" THEN
          PRINT "DORIAN: " + current's instance count
        ELSE IF current's word is "experimental" THEN
          PRINT "EXPERIMENTAL: " + current's instance count
        ELSE IF current's word is "magnetic" THEN
          PRINT "MAGNETIC: " + current's instance count
        END IF
        current = current's next
      END WHILE
    END FUNCTION

  FUNCTION printList()
```

```
    BEGIN
      current = head
      WHILE current is not null
        PRINT current's word + " " + current's instance count
        current = current's next
      END WHILE
    END FUNCTION

  // Other helper functions and member variable access methods go here

END CLASS
```

PSEUDOCODE FOR LINKED-LIST-NODE CLASS

```
CLASS LinkedListNode
  PRIVATE String word
  PRIVATE Integer instanceCount
  PRIVATE LinkedListNode next

  FUNCTION LinkedListNode(word: String)
    BEGIN
      // Constructor for the LinkedListNode class
      this.word = word
      this.instanceCount = 1 // Set to 1 as we are inserting a new word
      this.next = null
    END FUNCTION

  FUNCTION getWord() -> String
    BEGIN
      RETURN word
    END FUNCTION

  FUNCTION setWord(word: String)
    BEGIN
      this.word = word
    END FUNCTION

  FUNCTION getInstanceCount() -> Integer
    BEGIN
      RETURN instanceCount
    END FUNCTION

  FUNCTION setInstanceCount(instanceCount: Integer)
    BEGIN
      this.instanceCount = instanceCount
    END FUNCTION

  FUNCTION getNext() -> LinkedListNode
    BEGIN
      RETURN next
    END FUNCTION
```

```
  FUNCTION setNext(next: LinkedListNode)
    BEGIN
      this.next = next
    END FUNCTION

  FUNCTION incrementInstanceCount()
    BEGIN
      instanceCount = instanceCount + 1
    END FUNCTION

END CLASS
```