```java
package FileParser;

import List.LNode;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

/**
 * FileToList class is a class written to parse a supplied String input path to a text file into a
 * linear link LNode list. All special characters (and subsequent blank words) will be removed. A word
 * can be a number, and they are case-sensitive.
 *
 * ****************** Valid Input ******************
 *
 * input        A String path to a text input file.
 *
 * *************** Public Operations *****************
 *
 * N/A
 *
 * ***************** Global Variables *****************
 *
 * data         Public variable for header LNode pointer.
 *
 * @author Matt Laidman (5199807)
 * @version 1.1 (November 17, 2014)
 *      Added more special characters
 */
public class FileToList {

    public LNode data;                                          // Public header List.LNode pointer

    /**
     * Public constructor to call methods to parse the input file, and clean the words.
     *
     * @param input     The input String path to the file.
     */

    public FileToList ( String input ) {

        data = noBlanks(cleanWords(getWords(getFile(input))));
    }

    /**
     * Private noBlanks function removes any LNodes with blank String keys that may be in the list after
     * removing special characters.
     *
     * @param words     The list of words.
     * @return          THe list of words with no blank words.
     */

    private LNode noBlanks (LNode words) {

        LNode prev = null;
        LNode pres = words;

        while (pres != null) {
            if (pres.key.equals("") && prev == null && pres.next != null) {
                words = pres.next;                              // If first item in list, move pointer
                pres = pres.next;                               // forward one
            } else if (pres.key.equals("") && prev != null && pres.next != null) {
                prev.next = pres.next;                          // If internal List.LNode, point around
                pres = pres.next;
            } else if (pres.key.equals("") && prev != null && pres.next == null) {
                prev.next = null;                               // If last List.LNode, point to null ins
            } else {
                prev = pres;
                pres = pres.next;
            }
        }
```

```
    return words;
}

/**
 * Private cleanWords function calls the noSpecialChars function with each word in the list.
 *
 * @param words     The list of words.
 * @return          The cleaned list of words.
 */

private LNode cleanWords (LNode words) {

    LNode ptr = words;

    if(ptr == null) {                                       // If empty list, throw exception
        throw new FileToListException("No data given in file.");
    } else {
        while (ptr != null){                                // For each word, call noSpecialChars
            ptr.key = noSpecialChars(ptr.key);
            ptr = ptr.next;
        }

    }
    return words;                                           // Return cleaned words list
}

/**
 * Private noSpecialChars function removes any special characters in the given string. A special
 * character is defined by the validChar function.
 *
 * @param word      The string to remove characters from.
 * @return          The word with no special characters.
 */

private String noSpecialChars (String word) {

    String tWord = "";

    for (char c : word.toCharArray()) {                     // For each char in word
        if (validChar(c)) {                                 // If char is valid char
            tWord = tWord + c;                              // Append to new word
        }
    }
    return tWord;
}

/**
 * Private validChar function checks a given char against the invalids array. If the character is in
 * the array; false is returned, otherwise true is returned.
 *
 * Invalid characters are:
 *
 * . , - ( ) ;
 *
 * @param c     The character to check.
 * @return      True if character is valid, false otherwise.
 */

private boolean validChar (char c) {

    char[] invalids = {'.', ',', '-', '(', ')', ';', '?', '%'};

    for (char i : invalids) {                               // For each char in invalids array
        if (c == i) {                                       // If c is equal to invalid char
            return false;                                   // Return false
        }
    }
    return true;                                            // Otherwise return true.
}

/**
```

```
     * Private getWords function uses given Scanner's next function to read in each word in the file
     * to an LNode in the list.
     *
     * @param fileS     The Scanner of the text file.
     * @return          The list of words.
     */

    private LNode getWords (Scanner fileS) {

        LNode words = new LNode(fileS.next());                  // Read in first word
        LNode ptr = words;

        while (fileS.hasNext()) {                               // While there are words to read in
            ptr.next = new LNode(fileS.next());                 // Set next node to word
            ptr = ptr.next;
        }

        return words;
    }

    /**
     * Private getFile function checks if the supplied String input path is a valid text file and creates
     * a new Scanner from it.
     *
     * @param input     The input String path.
     * @return          The Scanner of the file.
     */

    private Scanner getFile (String input) {

        Scanner fileS;

        try {
            fileS = new Scanner(new File(input));               // Try creating Scanner
        } catch (FileNotFoundException e) {                     // Throw exception if bad file or path
            throw new FileToListException("File not found at: \"" + input + "\"");
        }
        return fileS;                                           // Return Scanner
    }
}
```