

**TEAM MOCHA**

**CECS 429 – Team Project Design**

**Prepared by: John**

**Reviewed & Revised by: Chris, Zhen**

**CLASS NAME:** Program

**AUTHOR(S):** Chris,John, and Zhen

**DESCRIPTION:**

Program plays host to the main function. It is also where all the other classes are called and executed.

**PUBLIC METHODS:**

**buildDictionaries** – accepts a String argument *filename* that is the name of the file to create the dictionaries for. It calls all of the classes found below in order to create the two dictionaries (type and term).

**displayMenu –** is our command-line user interface. It displays the items that the program does, and then prompts the user for input.

**DEVELOPERS NOTES:**

We initially divided parts of the program and started coding individually. We quickly found ourselves going in different directions, essentially creating different versions of the program. Having realized this, we decided to all sit down in the lab and create the structure of the program together. Our main focus was to make sure our separate classes worked together. We used Zhen’s code as base for the main program, evaluated each line and, together, made changes to reflect the ideal structure we have for the project. We decided to have *buildDictionaries* call all the methods of the different classes we made. This way it is easier to keep track of edits as well as fixing error in our code. John created a temporary command-line interface for the program demo (*displayMenu*) until Chris finishes the GUI components of the program.

**CLASS NAME:** Porterstem

**AUTHOR(S):** John, Zhen

**DESCRIPTION:**

Porterstem is the implementation of the Porter Stemmer Algorithm. It accepts an argument *token* and, after going through all the appropriate steps, returns the term equivalent of that token.

**PUBLIC METHOD:**

**ApplyPorterStem** – implements the stemming algorithm. It accepts an argument *token*, calls the private method *removeSuffix* to process the word, and then returns its term equivalent.

**DEVELOPER NOTES:**

Zhen wrote the first version of the stemmer. John did the testing and corrected the errors from the first version.

**CLASS NAME:** IOConnection

**AUTHOR(S):** Zhen

**DESCRIPTION:**

The IOConnection class opens the file containing the documents to be searched.

**PUBLIC METHODS:**

**openFile** – accepts an argument String that is the name of the file. It checks if the file exists. If it does, it returns an instance of FileReader. Otherwise, it prints an error messages and terminates the program.

**CLASS NAME:** Dictionary

**AUTHOR(S):** John, Zhen

**DESCRIPTION:**

The Dictionary class creates both the Type and Term dictionary. It keeps a list of all the types (*typeDict*) and terms (*termDict*) found in all the documents. *typeDict* is type Hashtable<String, String> where key = type and value = term. *termDict* is type Hashtable<String, Hashtable<Integer, ArrayList<Integer>>> where key = term and value = *postings*. *Postings* consist of *docID* (key) and a list of positions the term appeared in the particular document (value).

**PUBLIC METHODS:**

**addType –** accepts two String arguments, *type* and *term*. It checks if *type* is a new type. If it is, it appends the type and its corresponding term to the end of the list.

**addTerm –** accept a String argument *term* and two int arguments, *docID* and *posID*. It checks if *term* is a new term. If it is, it creates a new postings list with the corresponding *docID* and *posID*. Otherwise, it updates the postings list with the latest position (*posID*) in which the term appeared.

**displayTypes –** display all the types in the type dictionary.

**displayTerms –** display all the terms in the term dictionary.

**DEVELOPER NOTES:**

We initially divided parts of the program and started coding individually. We quickly found ourselves going in different directions, essentially creating different versions of the program. Having realized this, we decided to all sit down in the lab and create the structure of the program together. The Dictionary class was one of the classes we worked on heavily as a group.

**CLASS NAME:** Parser

**AUTHOR(S):** John

**DESCRIPTION:**

The Parser class parses a document into tokens by removing punctuations and lowercasing each word.

**PUBLIC METHODS:**

**parse –** accepts a String argument *document* and splits it into tokens. It stores all the tokens of the document in a Vector. Each call to the parse method, clears the previous contents of the Vector.

**getTokens –** returns the list of tokens.

**CLASS NAME:** QueryEx

**AUTHOR(S):** Zhen

**DESCRIPTION: get** document **ID that matches the query string. The query string could be in wildcard form, as well as being in normal DNF form.**

**PUBLIC METHODS:**

**takeQuery**: take one disjunctive form, and return the result in terms of document ID.

makeQuery: calling takeQuery multiple times as necessary. OR the all the list return from takeQuery function to get final document ID matches whole query string.

wildCardQuery: doing the same thing as makeQuery function, except called when the query string is wildcard query.

**DEVELOPER NOTES:**

The makeQuery and wildCardQuery function is called by constructor, so when instantiate new object, the result is automatically generated.

**CLASS NAME:** OpertionFunc

**AUTHOR(S):** Zhen

**DESCRIPTION:**

**PUBLIC METHODS:**

**DEVELOPER NOTES:**

**CLASS NAME:** DocumentWindow

**AUTHOR(S):** Chris

**DESCRIPTION:**

DocumentWindow encapsulates the Swing components that make up the window for presenting the user with a document matching the search query.

**PUBLIC METHODS:**

run() - called by SwingUtilites.invokeLater to show the window.

**DEVELOPER NOTES:**

DocumentWindow is invoked by double clicking on a list entry in the returned documentID's of a search query in MainWindow.

**CLASS NAME:** InvldxStatWindow

**AUTHOR(S):** Chris

**DESCRIPTION:**

InvIdxStatWindow's purpose is to allow the user to inspect the contents of the InvertedIndex. Contains a Jtree view for showing the contents of the inverted index.

**PUBLIC METHODS:**

run() - called by SwingUtilites.invokeLater to show the window.

**DEVELOPER NOTES:**

InvIdxStatWindow is invoked from the MainWindow.

**CLASS NAME:** MainWindow

**AUTHOR(S):** Chris

**DESCRIPTION:**

The main gui window of the program. Provies gui functionality for opening the windows for choosing a file to index, testing the porterstemmer, and showing the contents of the invertedIndex. Also provides the user interface for preform a search query and viewing a list of returned documents.

**PUBLIC METHODS:**

run() - called by SwingUitilites.invokeLater to show the window.

SetFile(String fileName) – used for setting the currently indexed file text of the MainWindow.

**DEVELOPER NOTES:**

When this window is closed the program exits.

**CLASS NAME:** Search

**AUTHOR(S):** Chris

**DESCRIPTION:**

A SwingWorker class that preforms the computation of a search query. This class runs in a seperate thread from the gui event thread, so that a search query wont block the gui from repainting itself or responding to user input. Asynchronously returns the search results to the main window once they are computed.

**PUBLIC METHODS:**

Search(String in) – constructor for passing in the query string to be processed.

doInBackground() - function invoked in a separate thread by swing for background processing of the search query.

**DEVELOPER NOTES:**

Results of Search are returned by way of swing PropertyChangeEvents to an event listener running in the main gui thread to allow for communication between Search and the MainWindow.