DS Final Project

Description:

Nowadays, more and more people appreciate independent music and become the fans of indie band. However, since independent music is not mainstream, there are no search engines which are suitable for those people to help them get the latest information. Therefore, we develop a search engine better than Google and other search engines for those people to search. The appearance of the GUI is as Fig. 1.

Figure 1. The graphical user interface
(Search.jsp)

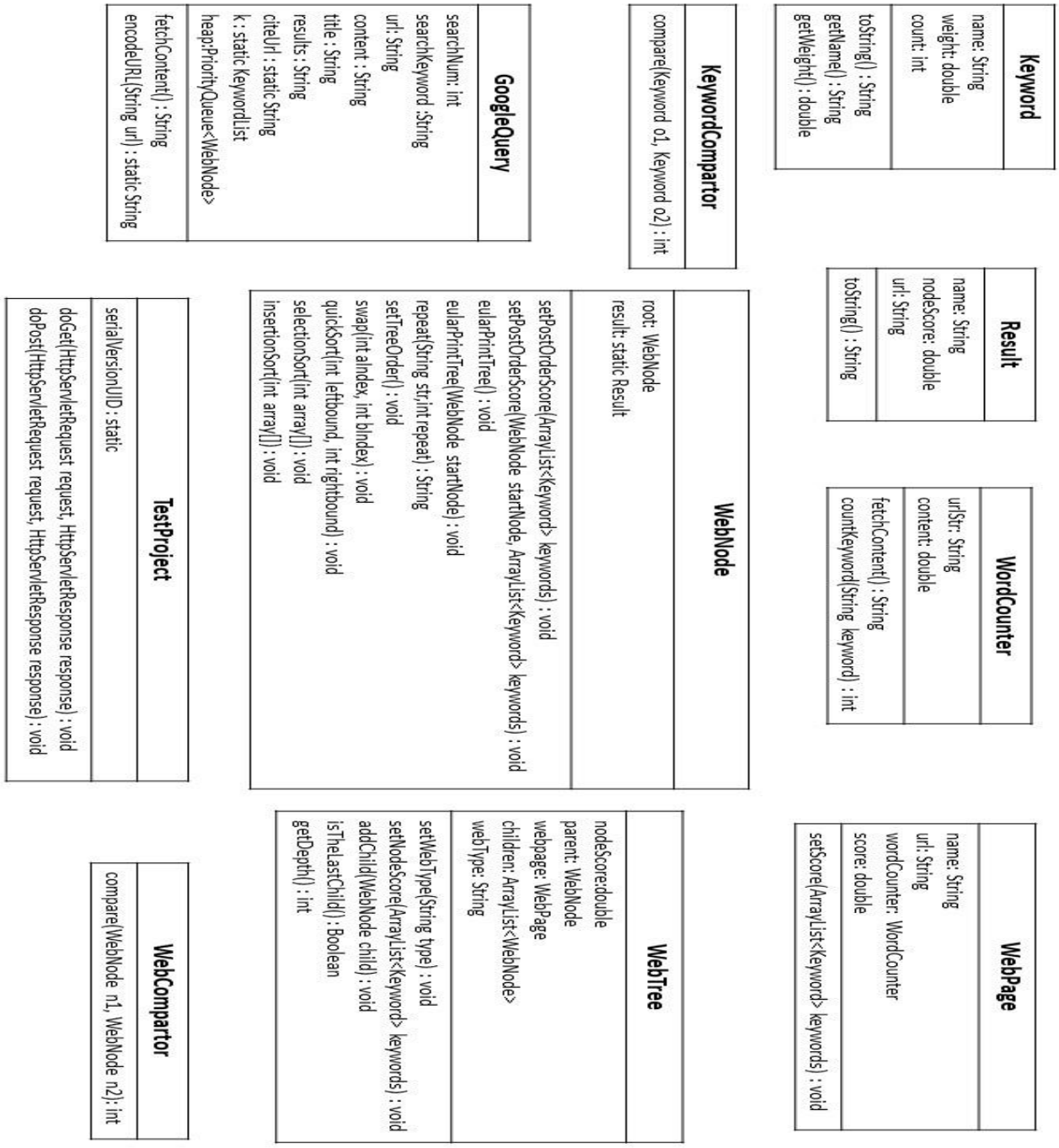

Figure 2. The UML Diagram

1. ***Keyword*** class

Keyword	
Modifier and type	**Method (or Variable) and description**
Instance variable	
String	name
double	weight
int	count
Constructor	
Keyword(String name, double weight) Enable to construct a *Student* object with given *name*, *weight*.	
Instance methods	
getters	getName(), getWeight()
String	toString() Return a String description of the keyword. **Sample output:** [海豚刑警 , 2]

2. ***KeywordComparator*** class

KeywordComparator implements Comparator<Keyword>	
Modifier and type	**Method (or Variable) and description**
Instance methods	
int	compare (Keyword o1, Keyword o2) Compare the count of the keywords and return (1, 0, -1) based on the result.

3. ***GoogleQuery*** class

GoogleQuery	
Modifier and type	**Method (or Variable) and description**
Instance variable	
int	searchNum
String	url
String	searchKeyword
String	content
String	title
String	results
static String	citeUrl
KeywordList	kLst
PriorityQueue<WebNode>	heap
long	startingTime
Constructor	
GoogleQuery(String searchKeyword) Construct a GoogleQuery object and set the input parameters for connecting the google server. Make sure to declare the url parameter with the related value below: ***this.url = "http://www.google.com/search?q=" + this.searchKeyword + "&oe=utf8&num=50";*** and initializes other parameters.	

Instance methods	
String	fetchContent () Use URLConnection and related methods to fetch the content we want.
HashMap<String, String>	query() Add founded websites and urls to Hashmap, and call the related methods to compute score of pages, print tree, sort and show results.
ArrayList<String>	hyper () Add founded websites and url to an ArrayList..
static String	encodeURL (String url) Encode the url which was founded.

4. ***Result*** class

Result	
Modifier and type	**Method (or Variable) and description**
Instance variable	
String	name
String	url
double	nodeScore
Constructor	
Result (String name, double nodeScore, String url) Enable to construct a Result object and instantiate the name, url and nodeScore.	
String	toString() Return a String description of the result.

5. ***KeywordList*** class

KeywordList	
Modifier and type	**Method (or Variable) and description**
Instance variable	
ArrayList<Result>	lst The execution for user to connect the input with the database.
Constructor	
KeywordList () Construct a KeywordList object and instantiate **ArrayList<Result>** lst.	
Instance methods	
ArrayList<Result>	getList () Return lst.
void	add(Result result) Add the result to lst.
void	sort() Use quickSort, bubbleSort, selectionSort, or insertionSort to sort items.
private void	quickSort(int leftbound, int rightbound) Implement quickSort.
private void	bubbleSort(int array[]) Implement bubbleSort.
private void	selectionSort(int array[]) Implement selectionSort.
private void	insertionSort(int array[]) Implement selectionSort.
void	swap(int a, int b) Swap the position of lst.
void	show() Show the result of sorting.

6. ***WebComparator*** class

WebComparator implements from Comparator<WebNode>	
Modifier and type	**Method (or Variable) and description**
Instance methods	
int	compare (WebNode n1, WebNode n2) Compare the nodeScore of the Webs and return a number based on the result.

7. ***WebNode*** class

WebNode	
Modifier and type	**Method (or Variable) and description**
Instance variable	
double	nodeScore
WebNode	parent
WebPage	webPage
ArrayList<WebNode>	children
String	webType
Constructor	
WebNode(WebPage webPage) Enable to construct a WebNode object and instantiate the webPage and children.	
void	setWebType (String type) Instantiate webType with given type.
void	setNodeScore (ArrayList<Keyword> keywords) Set the node score of keywords to arraylist.
void	addChild(WebNode child) Add the given child to children arraylist. Besides, set the child's parent is this.
boolean	isTheLastChild() Check whether it is the last child or not.
int	getDepth() Compute the depth of the node tree.

8. ***WebPage*** class

WebPage	
Modifier and type	**Method (or Variable) and description**
Instance variable	

String	url
String	name
WordCounter	wordCounter
double	score
Constructor	
WebPage(String url,String name) Enable to construct a WebNode object and instantiate the name, url and wordCounter. Besides, you also need to consider the UnsupportedEncodingException.	
void	setScore (ArrayList<Keyword> keywords) Set the score of keywords to arraylist.

9. ***WebTree*** class

WebTree	
Modifier and type	**Method (or Variable) and description**
Instance variable	
WebNode	root
static Result	result
Constructor	
WebTree(WebPage rootPage) Enable to construct a WebTree object and instantiate the root with given WebPage.	
void	setPostOrderScore(ArrayList<Keyword> keywords) Call the private void setPostOrderScore method.
private void	setPostOrderScore(WebNode startNode, ArrayList<Keyword> keywords) Implement setPostOrderScore.
void	eularPrintTree() Call the private void eularPrintTree method.
private void	eularPrintTree(WebNode startNode) Print the tree result include web pages and url.
private String	repeat(String str,int repeat) Return a string object.
void	setTreeOrder() Implement quickSort method.
private void	swap(int aIndex, int bIndex) Swap the position of root.children.

private void	quickSort(int leftbound, int rightbound) Implement quickSort.
private void	bubbleSort(int array[]) Implement bubbleSort.
private void	selectionSort(int array[]) Implement selectionSort.
private void	insertionSort(int array[]) Implement insertionSort.

10. ***WordCounter*** class

WordCounter	
Modifier and type	**Method (or Variable) and description**
Instance variable	
String	urlStr The website's nodeScore.
String	content The parent website.
Constructor	
WordCounter (String urlStr) Enable to construct a WordCounter object and instantiate the urlStr.	
String	fetchContent () Fetch the content of url
int	countKeyword(String keyword) Compute how many times does the keyword appear.

11. *TestProject*

TestProject
extends
HttpServlet

```
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.Map.Entry;
import java.util.Properties;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.*;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class TestProject
 */
@WebServlet("/TestProject")
public class TestProject extends HttpServlet {
	private static final long serialVersionUID = 1L;
```

```
/**
     * @see HttpServlet#HttpServlet()
     */
    public TestProject() {
        // Used as Main.java
       super();
    }

        /**
         * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
         */

        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

                // TODO Auto-generated method stub
                response.setCharacterEncoding("UTF-8");
                request.setCharacterEncoding("UTF-8");
                response.setContentType("text/html");

                int search = 20;
                if(request.getParameter("searchNum") != null) {
                        search = Integer.parseInt(request.getParameter("searchNum"));
                }

                // search.jsp
                if(request.getParameter("keyword")== null) {
                        String requestUri = request.getRequestURI();
                        request.setAttribute("requestUri", requestUri);
                        request.getRequestDispatcher("Search.jsp").forward(request, response);
                        return;
                }

                // googleitem.jsp
                GoogleQuery google;
                google = new GoogleQuery(request.getParameter("keyword"));
                HashMap<String, String> query = google.query();
                int index = query.size();
                String[][] s = new String[index][2];
                request.setAttribute("query", s);

                int num = 0;
                for(Entry<String, String> entry : query.entrySet()) {
                   String key = entry.getKey();
                   s[num][0] = key;
                   String val = entry.getValue();
                   s[num][1] = val;
                   num = num + 1;
                }
```

```
                KeywordList kLst = google.kLst;
                for(int i = 0 ; i < kLst.lst.size() ; i++) {
                        s[i][0] = kLst.lst.get(i).name;
                        s[i][1] = kLst.lst.get(i).url;
                }
                request.getRequestDispatcher("googleitem.jsp").forward(request, response);

        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
```