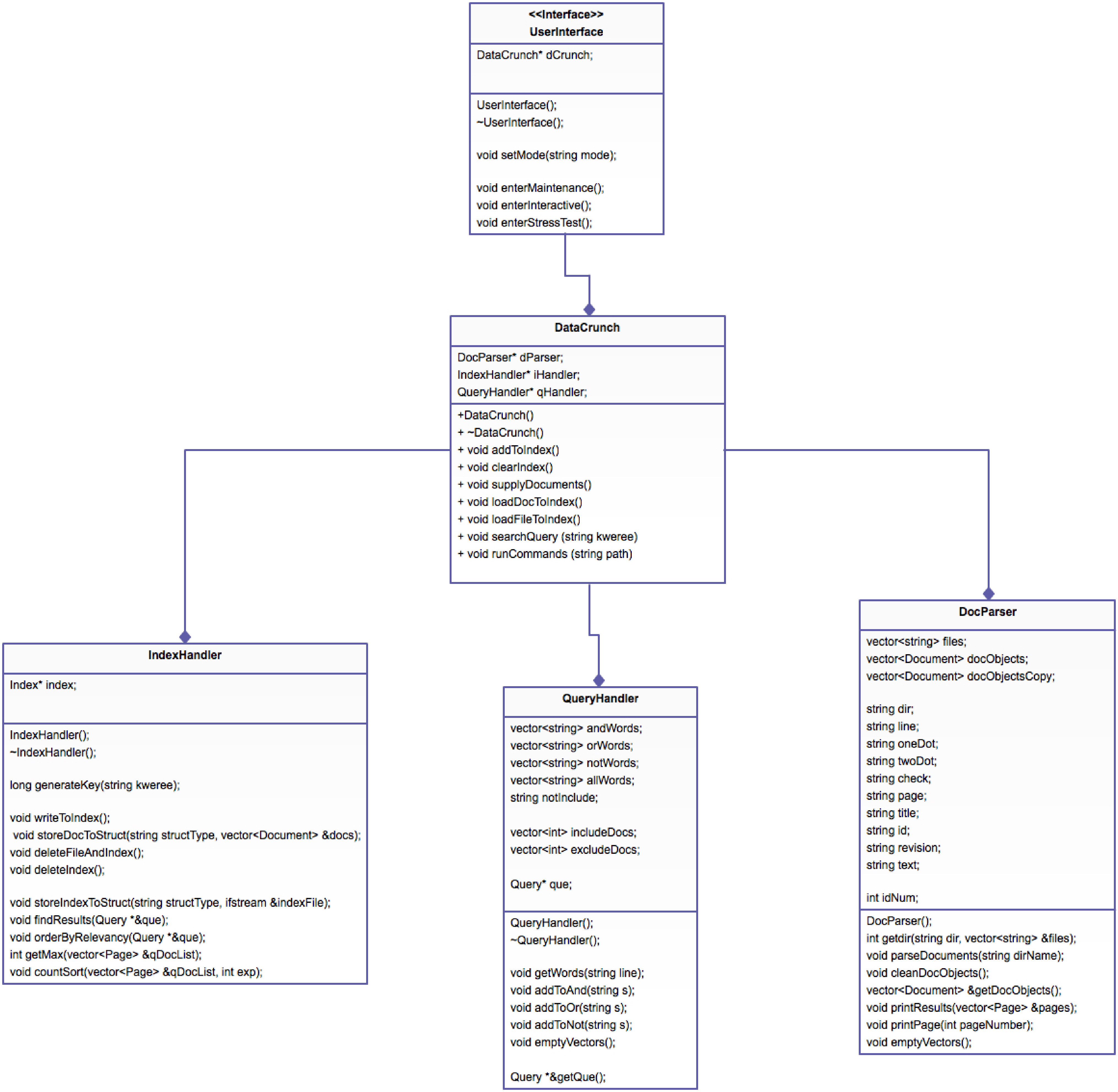
Team Zombie Coders
Design Document for Kwirē Search Engine

Team Memebers:
Emely Villeda-Principe
Aurora Havens



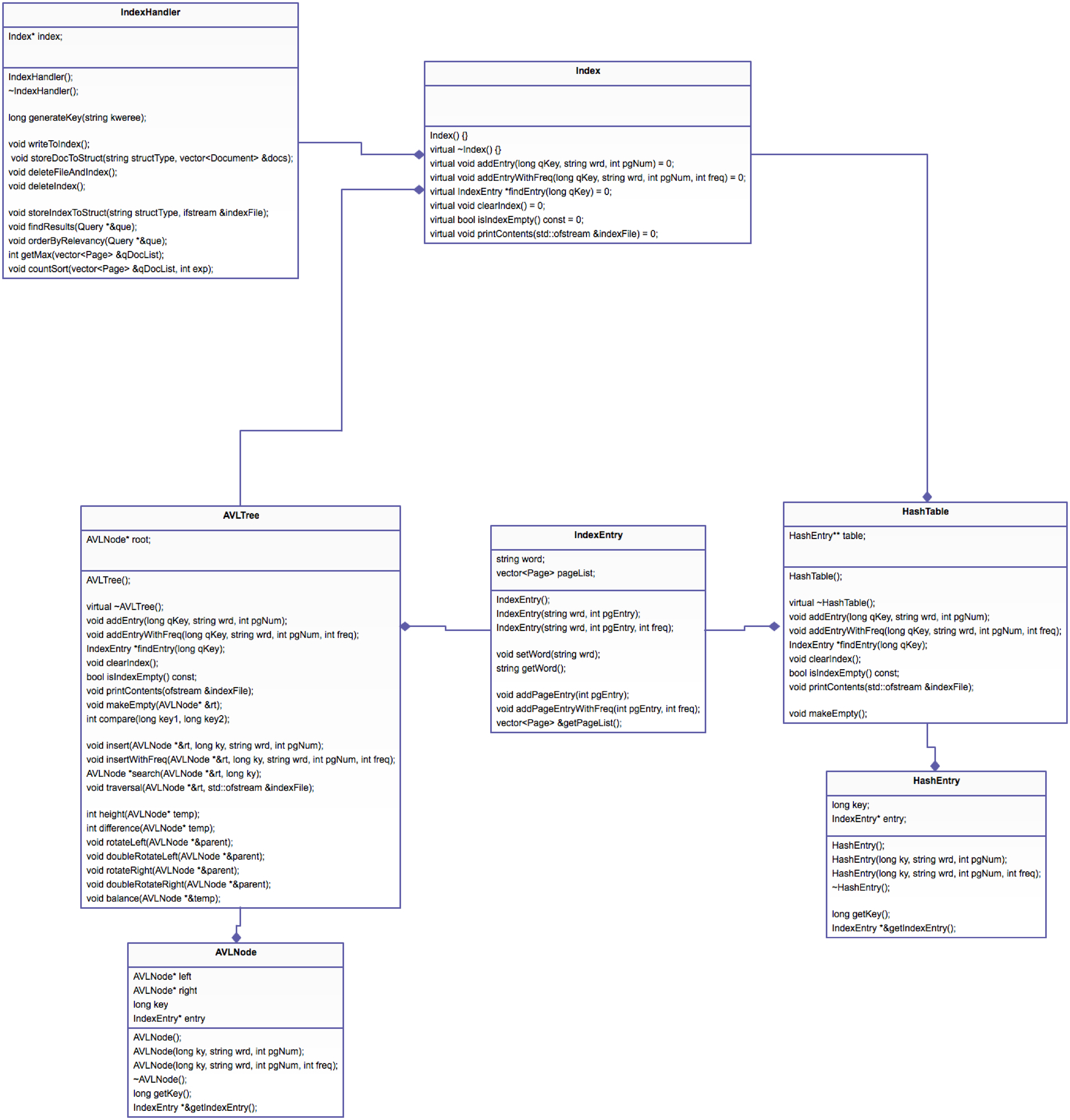
```
<<Interface>>
UserInterface
DataCrunch* dCrunch;

UserInterface();
~UserInterface();

void setMode(string mode);

void enterMaintenance();
void enterInteractive();
void enterStressTest();
```

```
DataCrunch
DocParser* dParser;
IndexHandler* iHandler;
QueryHandler* qHandler;

+DataCrunch()
+ ~DataCrunch()
+ void addToIndex()
+ void clearIndex()
+ void supplyDocuments()
+ void loadDocToIndex()
+ void loadFileToIndex()
+ void searchQuery (string kweree)
+ void runCommands (string path)
```

```
IndexHandler
Index* index;

IndexHandler();
~IndexHandler();

long generateKey(string kweree);

void writeToIndex();
 void storeDocToStruct(string structType, vector<Document> &docs);
void deleteFileAndIndex();
void deleteIndex();

void storeIndexToStruct(string structType, ifstream &indexFile);
void findResults(Query *&que);
void orderByRelevancy(Query *&que);
int getMax(vector<Page> &qDocList);
void countSort(vector<Page> &qDocList, int exp);
```

```
QueryHandler
vector<string> andWords;
vector<string> orWords;
vector<string> notWords;
vector<string> allWords;
string notInclude;

vector<int> includeDocs;
vector<int> excludeDocs;

Query* que;

QueryHandler();
~QueryHandler();

void getWords(string line);
void addToAnd(string s);
void addToOr(string s);
void addToNot(string s);
void emptyVectors();

Query *&getQue();
```

```
DocParser
vector<string> files;
vector<Document> docObjects;
vector<Document> docObjectsCopy;

string dir;
string line;
string oneDot;
string twoDot;
string check;
string page;
string title;
string id;
string revision;
string text;

int idNum;

DocParser();
int getdir(string dir, vector<string> &files);
void parseDocuments(string dirName);
void cleanDocObjects();
vector<Document> &getDocObjects();
void printResults(vector<Page> &pages);
void printPage(int pageNumber);
void emptyVectors();
```



```
IndexHandler
Index* index;

IndexHandler();
~IndexHandler();

long generateKey(string kweree);

void writeToIndex();
 void storeDocToStruct(string structType, vector<Document> &docs);
void deleteFileAndIndex();
void deleteIndex();

void storeIndexToStruct(string structType, ifstream &indexFile);
void findResults(Query *&que);
void orderByRelevancy(Query *&que);
int getMax(vector<Page> &qDocList);
void countSort(vector<Page> &qDocList, int exp);
```

```
Index

Index() {}
virtual ~Index() {}
virtual void addEntry(long qKey, string wrd, int pgNum) = 0;
virtual void addEntryWithFreq(long qKey, string wrd, int pgNum, int freq) = 0;
virtual IndexEntry *findEntry(long qKey) = 0;
virtual void clearIndex() = 0;
virtual bool isIndexEmpty() const = 0;
virtual void printContents(std::ofstream &indexFile) = 0;
```

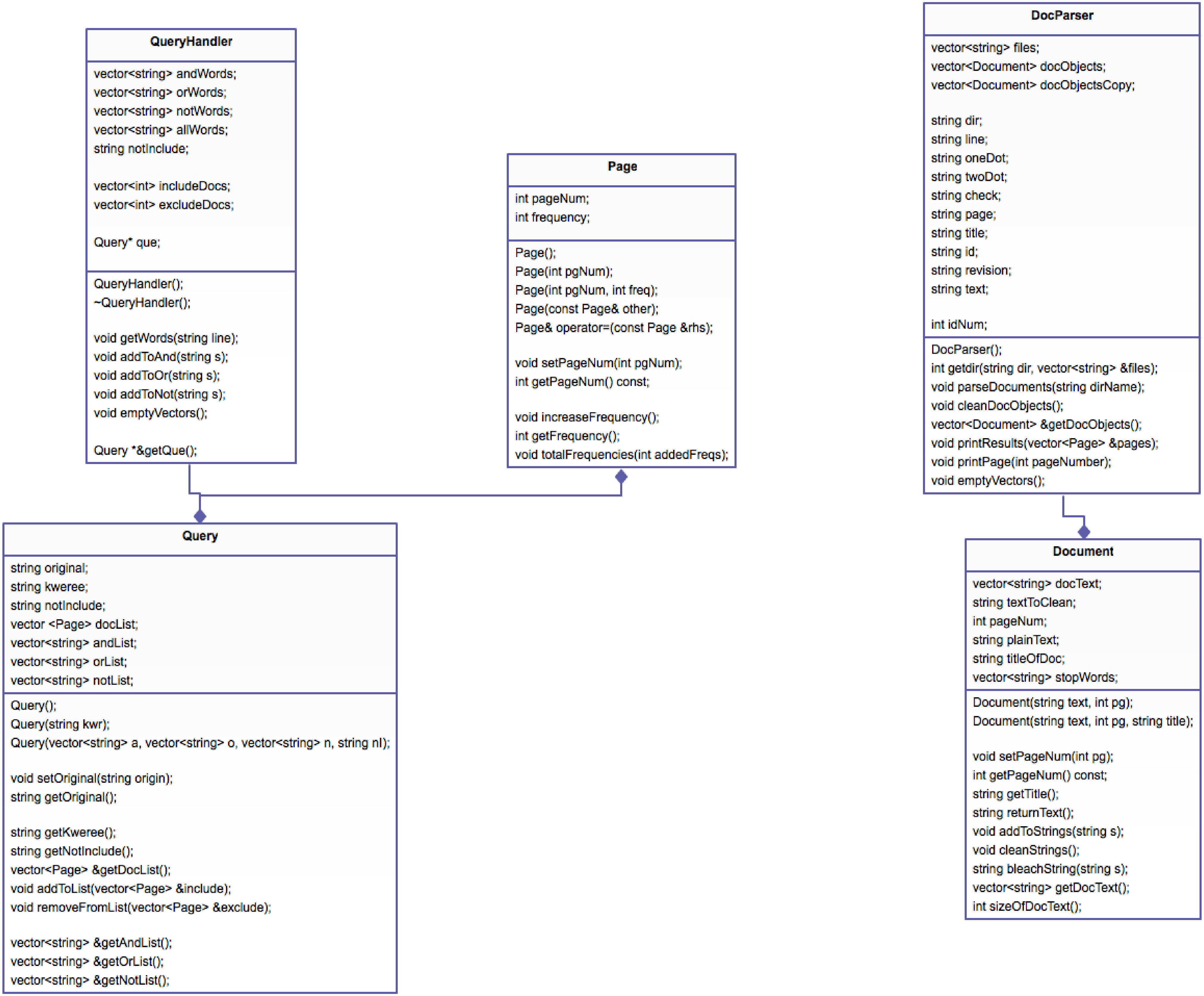
```
AVLTree
AVLNode* root;

AVLTree();

virtual ~AVLTree();
void addEntry(long qKey, string wrd, int pgNum);
void addEntryWithFreq(long qKey, string wrd, int pgNum, int freq);
IndexEntry *findEntry(long qKey);
void clearIndex();
bool isIndexEmpty() const;
void printContents(ofstream &indexFile);
void makeEmpty(AVLNode* &rt);
int compare(long key1, long key2);

void insert(AVLNode *&rt, long ky, string wrd, int pgNum);
void insertWithFreq(AVLNode *&rt, long ky, string wrd, int pgNum, int freq);
AVLNode *search(AVLNode *&rt, long ky);
void traversal(AVLNode *&rt, std::ofstream &indexFile);

int height(AVLNode* temp);
int difference(AVLNode* temp);
void rotateLeft(AVLNode *&parent);
void doubleRotateLeft(AVLNode *&parent);
void rotateRight(AVLNode *&parent);
void doubleRotateRight(AVLNode *&parent);
void balance(AVLNode *&temp);
```

```
IndexEntry
string word;
vector<Page> pageList;

IndexEntry();
IndexEntry(string wrd, int pgEntry);
IndexEntry(string wrd, int pgEntry, int freq);

void setWord(string wrd);
string getWord();

void addPageEntry(int pgEntry);
void addPageEntryWithFreq(int pgEntry, int freq);
vector<Page> &getPageList();
```

```
HashTable
HashEntry** table;

HashTable();

virtual ~HashTable();
void addEntry(long qKey, string wrd, int pgNum);
void addEntryWithFreq(long qKey, string wrd, int pgNum, int freq);
IndexEntry *findEntry(long qKey);
void clearIndex();
bool isIndexEmpty() const;
void printContents(std::ofstream &indexFile);

void makeEmpty();
```

```
HashEntry
long key;
IndexEntry* entry;

HashEntry();
HashEntry(long ky, string wrd, int pgNum);
HashEntry(long ky, string wrd, int pgNum, int freq);
~HashEntry();

long getKey();
IndexEntry *&getIndexEntry();
```

```
AVLNode
AVLNode* left
AVLNode* right
long key
IndexEntry* entry

AVLNode();
AVLNode(long ky, string wrd, int pgNum);
AVLNode(long ky, string wrd, int pgNum, int freq);
~AVLNode();
long getKey();
IndexEntry *&getIndexEntry();
```



QueryHandler

```
vector<string> andWords;
vector<string> orWords;
vector<string> notWords;
vector<string> allWords;
string notInclude;

vector<int> includeDocs;
vector<int> excludeDocs;

Query* que;
```

```
QueryHandler();
~QueryHandler();

void getWords(string line);
void addToAnd(string s);
void addToOr(string s);
void addToNot(string s);
void emptyVectors();

Query *&getQue();
```

Page

```
int pageNum;
int frequency;
```

```
Page();
Page(int pgNum);
Page(int pgNum, int freq);
Page(const Page& other);
Page& operator=(const Page &rhs);

void setPageNum(int pgNum);
int getPageNum() const;

void increaseFrequency();
int getFrequency();
void totalFrequencies(int addedFreqs);
```

DocParser

```
vector<string> files;
vector<Document> docObjects;
vector<Document> docObjectsCopy;

string dir;
string line;
string oneDot;
string twoDot;
string check;
string page;
string title;
string id;
string revision;
string text;

int idNum;
```

```
DocParser();
int getdir(string dir, vector<string> &files);
void parseDocuments(string dirName);
void cleanDocObjects();
vector<Document> &getDocObjects();
void printResults(vector<Page> &pages);
void printPage(int pageNumber);
void emptyVectors();
```

Query

```
string original;
string kweree;
string notInclude;
vector <Page> docList;
vector<string> andList;
vector<string> orList;
vector<string> notList;
```

```
Query();
Query(string kwr);
Query(vector<string> a, vector<string> o, vector<string> n, string nl);

void setOriginal(string origin);
string getOriginal();

string getKweree();
string getNotInclude();
vector<Page> &getDocList();
void addToList(vector<Page> &include);
void removeFromList(vector<Page> &exclude);

vector<string> &getAndList();
vector<string> &getOrList();
vector<string> &getNotList();
```

Document

```
vector<string> docText;
string textToClean;
int pageNum;
string plainText;
string titleOfDoc;
vector<string> stopWords;
```

```
Document(string text, int pg);
Document(string text, int pg, string title);

void setPageNum(int pg);
int getPageNum() const;
string getTitle();
string returnText();
void addToStrings(string s);
void cleanStrings();
string bleachString(string s);
vector<string> getDocText();
int sizeOfDocText();
```

Team Zombie Coders

Members: Aurora Havens and Emely Villeda-Principe

Class Roles and Responsibilities

- Class UserInterface **[Owner: Emely]**
  - Acts as the connector from main to DataCrunch in order to start the program
- Class DataCrunch **[Owner: Aurora]**
  - Accesses the 3 main processes that run the search engine (DocParser, IndexHandler, and QueryHandler)
  - Handles functions required for completing user requests
- Class DocParser **[Owner: Aurora (Including Document)]**
  - Takes in XML Documents to parse data to store relevant information
  - Responsible for retrieving specific data from the file and creating document objects with strings and pages numbers in side of them
- Class Document
  - Creates an object that will be stored out into the index so we can access the string and the documents that string is located in
- Class IndexHandler **[Owner: Emely (Including Index and its implementation classes)]**
  - Handles writing an index onto disk as well as storing the index data into an accessible data structure
  - Responsible for retrieving entries from the index and returning them to the user
- Class Index
  - Serves as the interface of the data structure that will hold the index data
  - Composed of index entries and the structure is dependent on the user
- Class IndexEntry
  - Serves as an entry containing the word and list of documents that include the term
  - For storing in a data structure that would allow for easy searching upon user request
- Class AVLTree
  - Serves as a potential data structure for storing index entries
  - Composed of nodes that are accessible to the user
- Class AVLNode
  - Node for the AVL Tree data structure
  - Holds IndexEntry object containing a string and a vector of document numbers
- Class HashTable
  - Serves as a potential data structure for storing index entries
  - Composed of HashElements that are accessible to the user
- Class HashElement
  - Element for the Hash Table data structure
  - Holds IndexEntry object containing a string and a vector of document numbers
- Class QueryHandler **[Owner: Aurora (Including Query)]**
  - Parses through query requests to determine what the IndexHandler must find
  - Responsible for interpreting the parts of a query (terms and Boolean operators)
- Class Query
  - Stores the parts of a query (terms associated with each Boolean operator)
  - Necessary for storing relevant documents once the IndexHandler finds matches

Team Zombie Coders

Members: Aurora Havens and Emely Villeda-Principe

Set of initial commands that are used in stress test mode to test the functionality of the program:

1. mmode
2. add
3. new
4. default
5. avl
6. return
7. imode
8. load
9. avl
10. search
11. main
12. return
13. mmode
14. clear
15. add
16. new
17. default
18. hash
19. return
20. imode