How to program with KEngine

Technical Report

Duc Minh Le
duclm@hanu.edu.vn
Faculty of IT, Hanoi University

1. Introduction

This report briefly describes how to use the KEngine API to write a program. The common tasks are the following:

1. initialise an Engine (Section 2)
2. initialise the document collection of the engine (Section 3)
3. search for documents using a query that contains one keyword (Section 4)
4. refine search query by incrementally adding more keywords (Section 5)

2. Initialise an Engine

Use the default constructor of the `Engine` class to initialise an Engine object.

```
Engine eng = new Engine();
```

3. Initialise the document collection

When an `Engine` object is first created, its document collection is empty. To initialise it, you need to first prepare at least one folder on your hard disk that contains some HTML documents. You then use the method `addDocs` to instruct the `Engine` object to load and parse the documents from this folder.

Suppose the absolute path to the document folder is `C:\kengine\docscol`. Lines 1-4 in Listing 1 initialise the full file-based URL to this directory. Note that the file separator character on the Windows OS must be escaped in Java code as "\\". However, if you are using Linux then its file separator ("/") is not a special character and thus needs not be escaped.

Lines 8-12 of the listing instruct the `Engine` object to load and parse documents from the folder.

Listing 1: How to initialise the document collection

```
1   String sep = File.separator;
2   String dirUrlPrefix = "file:" + sep + sep;
3   String docDirPath = "C:\\kengine\\docscol";
4   String docDirUrl = dirUrlPrefix + docDirPath;
5
6   // add docs from the data directory
```

```
System.out.println("Adding docs from URL: " + docDirUrl);
try {
  eng.addDocs(docDirUrl);
} catch (Exception e) {
  System.err.println("Error: " + e.getMessage());
}
```

4. Ask a single-keyword query

Once the Engine object has been initialised with some documents, you can start searching by formulating keyword queries. Line 4 in Listing 2 below shows how you can use the method `queryFirst` to ask a query with the first keyword.

The method `printMatches`, which is invoked at line 5, is displayed in Listing 3. It basically print out the matches (if any) of the query.

Listing 2: How to search for documents using keywords

```
1   try {
2     String key = "sinh";
3     System.out.println("\nQuerying using keyword: " + key);
4     Query q = eng.queryFirst(key);
5     printMatches(q);
6
7     key = "vien";
8     System.out.println("Refine query with keyword: " + key);
9     q = eng.queryMore(key);
10    printMatches(q);
11  } catch (Exception e) {
12    System.err.println("Error: " + e.getMessage());
13  }
```

Listing 3: Method `printMatches`

```
/**
 * @effects
 *  if q.matches is empty
 *    print "no matches"
 *  else
 *    print each match in q.matches (one per line)
 */
private static void printMatches(Query q) {
  // print matches
  if (q.size() == 0) {
    System.out.println("No matches");
```

```java
  } else {
    System.out.println("Matches");
    for (int di = 0; di < q.size(); di++) {
      Doc d = q.fetch(di);
      System.out.println((di + 1) + ": " + d.title());
    }
  }
}
```

5. Refine a query with more keywords

Once you've asked a single-keyword query, you can refine the query with more keywords. Line 9 of Listing 2 shows how to use the method `queryMore` to refine an existing query with one more keyword at a time.