UNIVERSITÉ D'AVIGNON ET DES PAYS DE VAUCLUSE

M2 ILSEN – 2019/20

UE Ingénierie du document et de l'information

UCE Indexation & recherche

Vincent Labatut

TP 1 | Correction

1 Tokénisation basique

Exercice 1

```
public int compareTo(Token token)
{ int result = type.compareTo(token.type);
  if(result==0)
     result = docId - token.docId;
  return result;
}
```

Exercice 2

```
public boolean equals(Object o)
{ boolean result = false;
  if(o instanceof Token)
  {  Token token = (Token) o;
     result = compareTo(token) == 0;
  }
  return result;
}

public String toString()
{ String result = "(" + type + ", " + docId + ")";
  return result;
}
```

Exercice 3

```
public List<String> tokenizeString(String string)
{ List<String> result = new LinkedList<String>();

  // on segmente la chaîne
  String temp[] = string.split("[^\\pL\\pN]");
  // on traite chaque segment
  for(String tmp: temp)
  {  // si le segment traité n'est pas vide
     if(!tmp.isEmpty())
        // on le rajoute àla liste résultat
        result.add(tmp);
  }

  return result;
}
```

Exercice 4

```
public void tokenizeDocument(File document, int docId, List<Token> tokens) throws
    UnsupportedEncodingException, FileNotFoundException
{ // on ouvre le flux
  FileInputStream fis = new FileInputStream(document);
  InputStreamReader isr = new InputStreamReader(fis,"UTF-8");
```

```
    Scanner scanner = new Scanner(isr);

    // on traite chaque ligne une par une
    while(scanner.hasNextLine())
    {  String line = scanner.nextLine();
       List<String> types = tokenizeString(line);
       for(String type: types)
       {  // on crée le token approprié
          Token token = new Token(type, docId);
          tokens.add(token);
       }
    }

    // on ferme le flux
    scanner.close();
 }
```

Exercice 5

```
 public int tokenizeCorpus(List<Token> tokens) throws UnsupportedEncodingException,
     FileNotFoundException
 { // on récupère les fichiers contenus dans le dossier
   String folder = FileTools.getCorpusFolder();
   File fFolder = new File(folder);
   String fileNames[] = fFolder.list();
   // note : le tri d'objets File peut varier d'un système àl'autre
   // mais pas le tri de String.
   Arrays.sort(fileNames);

   // nombre de documents dans la collection
   int result = fileNames.length;

   // on les traite un par un
   int docId = 0;
   for(String fileName: fileNames)
   {  File file = new File(folder+File.separator+fileName);
      tokenizeDocument(file, docId, tokens);
      docId++;
   }

   return result;
 }
```

2 Normalisation basique

Exercice 6

```
 public String normalizeType(String string)
 { String result = string;

   // on passe en minuscules
   result = result.toLowerCase();

   // on supprime les signes diacritiques
   result = java.text.Normalizer.normalize(
      result, Form.NFD).replaceAll("\\p{InCombiningDiacriticalMarks}+", "");

   // on renvoie null en cas de chaîne vide
   if(result.isEmpty())
      result = null;
```

```
14
15      return result;
16    }
```

Exercice 7

```
1   public void normalizeTokens(List<Token> tokens)
2   { // on traite chaque token séparément
3     Iterator<Token> it = tokens.iterator();
4     while(it.hasNext())
5     {  // on récupère le token
6        Token token = it.next();
7        // on normalise le type
8        String type = token.getType();
9        String term = normalizeType(type);
10       // si aucun terme n'est renvoyé, on supprime le token
11       if(term==null)
12          it.remove();
13       else
14          token.setType(term);
15    }
16  }
```

3 Fichier inverse simple

Exercice 8

```
1   public int compareTo(Posting posting)
2   { int result = docId - posting.docId;
3     return result;
4   }
5
6   public String toString()
7   { String result = Integer.toString(docId);
8     return result;
9   }
10
11  public boolean equals(Object o)
12  { boolean result = false;
13    if(o instanceof Posting)
14    {  Posting posting = (Posting) o;
15       result = compareTo(posting) == 0;
16    }
17    return result;
18  }
```

Exercice 9

```
1   public IndexEntry(String term)
2   { this.term = term;
3     this.postings = new ArrayList<Posting>();
4     this.frequency = 0;
5   }
6
7   public int compareTo(IndexEntry entry)
8   { int result = term.compareTo(entry.term);
9     return result;
10  }
11
12  public String toString()
13  { String result = "<" + term + " [" + frequency + "]" + " (";
```

```
      for(Posting posting: postings)
         result = result + " " + posting;
      result = result + " )>";
      return result;
    }

    public boolean equals(Object o)
    { boolean result = false;
      if(o instanceof IndexEntry)
      {  IndexEntry entry = (IndexEntry) o;
         result = compareTo(entry) == 0;
      }
      return result;
    }
```

Exercice 10

Constructeur de la classe-fille `ArrayIndex` :

```
public class ArrayIndex extends AbstractIndex
{  public ArrayIndex(int size)
   {  data = new IndexEntry[size];
   }
}
```

Constructeur de la classe-fille `HashIndex` :

```
public class HashIndex extends AbstractIndex
{  public HashIndex(int size)
   {  data = new HashMap<String,IndexEntry>(size);
   }
}
```

Constructeur de la classe-fille `TreeIndex` :

```
public class TreeIndex extends AbstractIndex
{  public TreeIndex()
   {  data = new TreeMap<String,IndexEntry>();
   }
}
```

Exercice 11

Dans la classe-fille `ArrayIndex` :

```
   public void print()
   {  for(int i=0;i<data.length;i++)
         System.out.println(i+".\t"+data[i]);
   }
```

Dans les classes-filles `HashIndex` et `TreeIndex` :

```
   public void print()
   {  int i = 0;
      for(IndexEntry indexEntry: data.values())
      {  System.out.println(i+".\t"+indexEntry);
         i++;
      }
   }
```

Exercice 12

Dans la classe-fille `ArrayIndex` :

```
   public void addEntry(IndexEntry indexEntry, int rank)
   {  data[rank] = indexEntry;
   }
```

Dans les classes-filles `HashIndex` et `TreeIndex` :

```
public void addEntry(IndexEntry indexEntry, int rank)
{  String key = indexEntry.getTerm();
   data.put(key,indexEntry);
}
```

Exercice 13

Dans la classe-fille `ArrayIndex`, une version avec la recherche dichotomique implémentée à la main :

```
public IndexEntry getEntry(String term)
{  IndexEntry result = null;

   if(data.length>0)
   {  // on effectue une recherche dichotomique classique
      int s=0,e=data.length-1;
      do
      {  int m = (s+e)/2;
         String t = data[m].term;
         int comp = term.compareTo(t);
         // term == t
         if(comp==0)
            result = data[m];
         // term > t
         else if(comp>0)
            s = m + 1;
         // term < t
         else if(comp<0)
            e = m - 1;
      }
      while(result==null && s<=e);
   }

   return result;
}
```

Alternativement, une version utilisant l'implémentation de `Arrays` :

```
public IndexEntry getEntry(String term)
{  IndexEntry result = null;

   // on a besoin d'un objet IndexEntry bidon
   IndexEntry entry = new IndexEntry(term);
   // on applique la méthode du JDK
   int pos = Arrays.binarySearch(data, entry);
   // on récupère l'objet adéquat
   if(pos>=0)
      result = data[pos];
   return result;
}
```

Dans les classes-filles `HashIndex` et `TreeIndex` :

```
public IndexEntry getEntry(String term)
{  IndexEntry result = data.get(term);
   return result;
}
```

Exercice 14

Dans la classe-fille `ArrayIndex` :

```
public int getSize()
```

```
2   {  int result = data.length;
3      return result;
4   }
```

Dans les classes-filles `HashIndex` et `TreeIndex` :

```
1   public int getSize()
2   {  int result = data.size();
3      return result;
4   }
```