```java
package st;

import java.util.ArrayList;

public class TemplateEngine {

    private static final Character TEMPLATE_START_PREFIX = '$';
    private static final Character TEMPLATE_START = '{';
    private static final Character TEMPLATE_END = '}';

    private static final String MM_KEEP = "keep-unmatched";
    private static final String MM_DELETE = "delete-unmatched";

    public TemplateEngine(){

    }

    public String evaluate(String templateString, EntryMap entryMap, String matchingMode){
        if (!isEvaluationPossible(templateString, entryMap)){
            return templateString;
        }
        if (!isMatchingModeValid(matchingMode)){
            matchingMode = MM_DELETE;
        }

        HashSet<Template> templates = identifyTemplates(templateString);

        ArrayList<Template> sortedTemplates = sortTemplates(templates);

        Result result = instantiate(templateString, sortedTemplates, entryMap.getEntries(), matchingMode);

        return result.getInstancedString();
    }

    private Boolean isEvaluationPossible(String templateString, EntryMap entryMap){
        if (templateString == null){
            return Boolean.FALSE;
        }
        if (templateString.isEmpty()) {
            return Boolean.FALSE;
        }
        if (entryMap == null){
            return Boolean.FALSE;
        }
        return Boolean.TRUE;
    }

    private Boolean isMatchingModeValid(String matchingMode){
        if (matchingMode == null){
            return Boolean.FALSE;
```

```
53          }
54          if (matchingMode.equals(MM_KEEP)){
55              return Boolean.TRUE;
56          }
57          if (matchingMode.equals(MM_DELETE)){
58              return Boolean.TRUE;
59          }
60          return Boolean.FALSE;
61      }
62
63      private HashSet<Template> identifyTemplates(String templateString){
64          HashSet<Template> templates = new HashSet<>();
65          Stack<Integer> templateCandidates = new Stack<>();
66          Integer charIndex = 0;
67          Boolean underSequence = Boolean.FALSE;
68          while (charIndex < templateString.length()){
69              if (Character.compare(templateString.charAt(charIndex),
    TEMPLATE_START_PREFIX) == 0){
70                  underSequence = Boolean.TRUE;
71                  charIndex++;
72                  continue;
73              }
74              if (Character.compare(templateString.charAt(charIndex),
    TEMPLATE_START) == 0){
75                  if(underSequence){
76                      templateCandidates.add(charIndex);
77                  }
78                  underSequence = Boolean.FALSE;
79                  charIndex++;
80                  continue;
81              }
82              if (Character.compare(templateString.charAt(charIndex),
    TEMPLATE_END) == 0){
83                  if (!templateCandidates.isEmpty()){
84                      Template template;
85                      Integer startIndex = templateCandidates.pop();
86                      if ((startIndex + 1) == charIndex){
87                          template = new Template(startIndex, charIndex,
    "");
88                      }
89                      else{
90                          template = new Template(startIndex, charIndex,
    templateString.substring(startIndex+1, charIndex));
91                      }
92                      templates.add(template);
93                  }
94                  underSequence = Boolean.FALSE;
95                  charIndex++;
96                  continue;
97              }
98              underSequence = Boolean.FALSE;
99              charIndex++;
100         }
```

```
101         return templates;
102     }
103 
104     private ArrayList<Template> sortTemplates(HashSet<Template> templates)
    {
105         ArrayList<Template> sortedTemplates = new ArrayList<>();
106         Template currentTemplate;
107         Integer minLength;
108         Integer startIndex;
109         while (!templates.isEmpty()) {
110             currentTemplate = null;
111             minLength = Integer.MAX_VALUE;
112             startIndex = Integer.MAX_VALUE;
113             for (Template current : templates){
114                 if (current.getContent().length() < minLength){
115                     currentTemplate = current;
116                     minLength = current.getContent().length();
117                     startIndex = current.getStartIndex();
118                 }
119                 else{
120                     if (current.getContent().length() == minLength){
121                         if (current.getStartIndex() < startIndex){
122                             currentTemplate = current;
123                             minLength = current.getContent().length();
124                             startIndex = current.getStartIndex();
125                         }
126                     }
127                 }
128             }
129             if (currentTemplate != null) {
130                 templates.remove(currentTemplate);
131                 sortedTemplates.add(currentTemplate);
132             }
133             else{
134                 throw new RuntimeException();
135             }
136         }
137         return sortedTemplates;
138     }
139 
140     private Result instantiate(String instancedString, ArrayList<Template>
    sortedTemplates, ArrayList<EntryMap.Entry> sortedEntries, String
    matchingMode){
141         Integer templatesReplaced = 0;
142         Boolean replaceHappened;
143         Template currentTemplate;
144         EntryMap.Entry currentEntry;
145         for (Integer i=0; i<sortedTemplates.size(); i++){
146             currentTemplate = sortedTemplates.get(i);
147             replaceHappened = Boolean.FALSE;
148             for(Integer j=0; j<sortedEntries.size(); j++){
149                 currentEntry = sortedEntries.get(j);
150                 if (isAMatch(currentTemplate, currentEntry)){
```

```
151                     instancedString = doReplace(instancedString,
    currentTemplate, i, currentEntry.getValue(), sortedTemplates);
152                     replaceHappened = Boolean.TRUE;
153                     break;
154                 }
155             }
156             if(replaceHappened){
157                 templatesReplaced ++;
158             }
159             else{
160                 if(matchingMode.equals(MM_DELETE)){
161                     instancedString = doReplace(instancedString,
    currentTemplate, i, "", sortedTemplates);
162                 }
163             }
164         }
165         return new Result(instancedString, templatesReplaced);
166     }
167 
168     private Boolean isAMatch(Template template, EntryMap.Entry entry){
169         String leftHandSide = template.getContent().replaceAll("\\s","");
170         String rightHandSide = entry.getPattern().replaceAll("\\s","");
171         if (entry.caseSensitive){
172             return leftHandSide.equals(rightHandSide);
173         }
174         else{
175             return leftHandSide.toLowerCase().equals
    (rightHandSide.toLowerCase());
176         }
177     }
178 
179     private String doReplace(String instancedString, Template
    currentTemplate, Integer currentTemplateIndex, String replaceValue,
    ArrayList<Template> sortedTemplates){
180         Integer diff = 3 + currentTemplate.getContent().length() -
    replaceValue.length();
181         String firstHalf;
182         String secondHalf;
183         if (currentTemplate.getStartIndex() == 1){
184             firstHalf = "";
185         }
186         else{
187             firstHalf = instancedString.substring(0,
    currentTemplate.getStartIndex()-1);
188         }
189         if (currentTemplate.getEndIndex() == instancedString.length()){
190             secondHalf = "";
191         }
192         else{
193             secondHalf = instancedString.substring
    (currentTemplate.getEndIndex()+1);
194         }
195 
```

```
196         StringBuilder builder = new StringBuilder();
197         builder.append(firstHalf);
198         builder.append(replaceValue);
199         builder.append(secondHalf);
200         String updatedInstancedString = builder.toString();
201 
202         Template temp = null;
203         for (int i=currentTemplateIndex+1; i<sortedTemplates.size(); i++){
204             temp = sortedTemplates.get(i);
205             if ((temp.getStartIndex() < currentTemplate.getStartIndex())
    && (temp.getEndIndex() > currentTemplate.getEndIndex()))
206             {
207                 sortedTemplates.get(i).setEndIndex(temp.getEndIndex() -
    diff);
208                 sortedTemplates.get(i).setContent
    (updatedInstancedString.substring(sortedTemplates.get(i).getStartIndex()
    +1, sortedTemplates.get(i).getEndIndex()));
209             }
210             else {
211                 if (temp.getStartIndex() > currentTemplate.getEndIndex())
    {
212                     sortedTemplates.get(i).setStartIndex
    (temp.getStartIndex() - diff);
213                     sortedTemplates.get(i).setEndIndex(temp.getEndIndex()
    - diff);
214                 }
215             }
216         }
217         return updatedInstancedString;
218     }
219 
220     class Template {
221         Integer startIndex;
222         Integer endIndex;
223         String content;
224 
225         Template(Integer startIndex, Integer endIndex, String content) {
226             this.startIndex = startIndex;
227             this.endIndex = endIndex;
228             this.content = content;
229         }
230 
231         public Integer getStartIndex() {
232             return startIndex;
233         }
234 
235         public Integer getEndIndex() {
236             return endIndex;
237         }
238 
239         public String getContent() {
240             return content;
241         }
```

```
242
243         public void setStartIndex(Integer startIndex) {
244             this.startIndex = startIndex;
245         }
246
247         public void setEndIndex(Integer endIndex) {
248             this.endIndex = endIndex;
249         }
250
251         public void setContent(String content) {
252             this.content = content;
253         }
254
255         @Override
256         public boolean equals(Object o) {
257             if (this == o) return true;
258             if (o == null || getClass() != o.getClass()) return false;
259
260             Template template = (Template) o;
261
262             if (getStartIndex() != null ? !getStartIndex().equals
    (template.getStartIndex()) : template.getStartIndex() != null)
263                 return false;
264             if (getEndIndex() != null ? !getEndIndex().equals
    (template.getEndIndex()) : template.getEndIndex() != null)
265                 return false;
266             return getContent() != null ? getContent().equals
    (template.getContent()) : template.getContent() == null;
267         }
268
269         @Override
270         public int hashCode() {
271             int result = getStartIndex() != null ? getStartIndex
    ().hashCode() : 0;
272             result = 31 * result + (getEndIndex() != null ? getEndIndex
    ().hashCode() : 0);
273             result = 31 * result + (getContent() != null ? getContent
    ().hashCode() : 0);
274             return result;
275         }
276     }
277
278     class Result{
279         String instancedString;
280         Integer templatesReplaced;
281
282         Result(String instancedString, Integer templatesReplaced) {
283             this.instancedString = instancedString;
284             this.templatesReplaced = templatesReplaced;
285         }
286
287         String getInstancedString() {
288             return instancedString;
```

```
289         }
290
291         Integer getTemplatesReplaced() {
292             return templatesReplaced;
293         }
294     }
295 }
296
```