## Mutations of StringUtils

```
1 import java.util.ArrayList;
   public class StringUtils {
       public static void main(String[] args){
5
8
10
       private static volatile char escape = 'e';
12
       public static char getEscape() {
13
           return escape;
14
16
       public static void setEscape(char escape) {
17
            StringUtils.escape = escape;
18
20
       public static String replaceString (String inputText,
           String pattern, String replacement, Character
           delimiter, boolean inside) throws RuntimeException
21
            if (!StringUtils.getMatchingStatus(inputText,
               pattern)){
22
               return inputText;
23
24
            StringBuilder sbInput = new StringBuilder (
               inputText);
25
            StringBuilder sbPattern = new StringBuilder (
               pattern);
            if(Character.compare(escape, '\\') == 0){
27
28
               throw new RuntimeException();
29
            }
```

```
31
            if(replacement = null){
32
                replacement = "";
33
35
            int charIndex = 0;
            1 \Delta int charIndex = 1;
37
            boolean underEscapeMode = false;
38
            boolean erased;
39
            boolean delimiterMode= StringUtils.
                getDelimiterMode(delimiter, inside);
40
            while (charIndex < sbPattern.length()){
            2\Delta while (charIndex <= sbPattern.length()){
41
42
            3\Delta while (charIndex > sbPattern.length()){
43
                if (underEscapeMode) {
                     underEscapeMode = false;
44
                     charIndex++;
45
                }
46
47
                else{
48
                     erased = false;
                     if (Character.compare(sbPattern.charAt(
49
                         charIndex), StringUtils.getEscape())
                         underEscapeMode = true;
50
51
                         sbPattern.deleteCharAt(charIndex);
                         erased = true;
52
53
                     if(delimiterMode && (Character.compare(
54
                         sbPattern.charAt(charIndex), delimiter
                         ) == 0) && !underEscapeMode) {
55
                     4\Delta if (delimiter Mode | | (Character.compare
                         (sbPattern.charAt(charIndex),
                         delimiter) == 0) && !underEscapeMode) {
56
                         sbPattern.deleteCharAt(charIndex);
57
                         erased = true;
58
                     if (!erased) {
59
60
                         charIndex++;
                     }
61
62
                }
63
            if(sbInput.length() < sbPattern.length()){</pre>
65
                return sbInput.toString();
66
67
            }
```

```
69
              if(sbInput.length() == sbPattern.length()){
70
                  if(sbInput.toString().equals(sbPattern.
                      toString()) && !inside){
71
                       return replacement;
72
                  }
                  else{
73
                       return sbInput.toString();
74
75
                  }
76
              }
              if (delimiterMode) {
78
79
                  ArrayList < Integer > startingPoints = new
                      ArrayList <>();
                  ArrayList < Integer > endingPoints = new
80
                      ArrayList <>();
81
                  boolean start = true;
                  for (int i = 0; i < sbInput.length(); i++){
82
                       Character currentChar = sbInput.charAt(i)
83
84
                       if (Character.compare (delimiter,
                           currentChar) = 0)
                           if(start) {
85
                                startingPoints.add(i);
86
                                start = false;
87
88
                           }
                            else{
89
90
                                endingPoints.add(i);
91
                                start = true;
92
                       }
93
94
                  if (endingPoints.isEmpty()){
95
                       if(inside){
96
                           return sbInput.toString();
97
98
                       else {
99
100
                            String Utils.doMatch(sbInput,
                               sbPattern, replacement, 0, sbInput
                               .length());
101
                           return sbInput.toString();
                       }
102
                  }
103
                  else{}
104
105
                       \mathbf{if} \, (\, starting \, Points \, . \, get \, (\, starting \, Points \, . \, size \,
                           ()-1) > endingPoints.get(endingPoints.
```

```
size()-1)
106
                          startingPoints.remove(startingPoints.
                              size()-1);
107
                     boolean replaceDone = false;
108
109
                     int oldLen;
                      if(inside){
110
                          for (int i=0; i<startingPoints.size()
111
                              ; i++){}
                              if(startingPoints.get(i)+1 <</pre>
112
                                  endingPoints.get(i)){
113
                                  oldLen = sbInput.length();
114
                                   if(doMatch(sbInput, sbPattern
                                      , replacement,
                                      startingPoints.get(i)+1,
                                      endingPoints.get(i))){
115
                                       replaceDone = true;
116
                                       updatePoints (
                                          startingPoints,
                                          endingPoints,
                                          startingPoints.get(i),
                                           sbInput.length() -
                                          oldLen);
117
                                  }
                              }
118
119
120
                          if (!replaceDone && (startingPoints.
                              get(0)+1 < endingPoints.get(
                              endingPoints.size()-1))
121
                              doMatch(sbInput, sbPattern,
                                  replacement, startingPoints.
                                  get(0)+1, endingPoints.get(
                                  endingPoints.size()-1);
122
                          }
123
124
                     else {
125
                          int startIndex;
126
                          int endIndex;
                          if(startingPoints.get(0) > 0){
127
128
                              startIndex = 0;
129
                              oldLen = sbInput.length();
130
                              if(doMatch(sbInput, sbPattern,
                                  replacement, 0, starting Points.
                                  get(0))) {
131
                                  replaceDone = true;
132
                                   updatePoints (startingPoints,
```

```
endingPoints, 0, sbInput.
                                       length() - oldLen);
133
                               }
134
135
                          else{
136
                               startIndex = endingPoints.get(0)
                                   +1;
137
                           if (endingPoints.get (endingPoints.size
138
                              ()-1)+1 < sbInput.length()){
139
                               endIndex = sbInput.length();
140
                               if(doMatch(sbInput, sbPattern,
                                   replacement, endingPoints.get(
                                   endingPoints.size()-1)+1,
                                   sbInput.length())) {
141
                                   replaceDone = true;
142
143
                          }
144
                          else {
145
                               endIndex = startingPoints.get(
                                   starting Points. size ()-1)-1;
146
                           for (int i=0; i < \text{endingPoints.size} () -1;
147
                               i++){}
148
                               if(endingPoints.get(i)+1 <
                                   startingPoints.get(i+1)){
149
                                   oldLen = sbInput.length();
150
                                   if(doMatch(sbInput, sbPattern)
                                       , replacement, ending Points.
                                       get(i)+1, startingPoints.
                                       get(i+1))){
151
                                        replaceDone = true;
152
                                        updatePoints (
                                           startingPoints,
                                           endingPoints,
                                           endingPoints.get(i),
                                           sbInput.length() -
                                           oldLen);
153
                                   }
154
                               }
155
                           if(!replaceDone && (startIndex <</pre>
156
                              endIndex)){
157
                               doMatch(sbInput, sbPattern,
                                   replacement, startIndex,
                                   endIndex);
```

```
158
                           }
159
160
                       return
                                sbInput.toString();
                  }
161
162
              }
163
              else{
                  String Utils.doMatch(sbInput, sbPattern,
164
                      replacement, 0, sbInput.length());
165
                  return sbInput.toString();
166
              }
167
         }
169
         private static boolean doMatch (StringBuilder input,
             StringBuilder pattern, String replace, Integer
             start, Integer end) {
170
              String sub = input.substring(start, end);
171
              String newSub = StringUtils.replaceAll(sub,
                  pattern.toString(), replace);
172
              if (sub.equals (newSub)) {
173
                  return false;
174
              }
175
              else {
176
                  input.replace(start, end, newSub);
177
                  return true;
178
              }
179
         }
181
         private static String replaceAll(String source,
             String from, String to) {
182
              StringBuilder builder = new StringBuilder (source)
183
              int index = builder.indexOf(from);
              while (index != -1)
184
185
                  builder.replace(index, index + from.length(),
186
187
                  index += to.length();
                  index = builder.indexOf(from, index);
188
189
190
             return builder.toString();
191
         }
         private static void updatePoints(ArrayList<Integer>
193
             startingPoints\;,\;\; ArrayList {<} Integer {>}\; endingPoints\;,
             \mathbf{int} \ \mathrm{index} \ , \ \mathbf{int} \ \mathrm{diff} \,) \, \{
194
              for(int i=0; i < startingPoints.size(); i++){
```

```
195
                 if(startingPoints.get(i) > index){
196
                     startingPoints.set(i, startingPoints.get(
                         i) + diff);
197
198
                 if(endingPoints.get(i) > index){
199
                     endingPoints.set(i, endingPoints.get(i) +
                          diff);
200
                 }
            }
201
202
        }
204
        private static Boolean getDelimiterMode(Character
            delimiterChar, Boolean insideFlag) throws
            RuntimeException {
205
             if(delimiterChar == null){
206
                 if(insideFlag){
207
                     throw new RuntimeException();
208
209
                 return false;
210
211
            return true;
212
        }
        private static boolean getMatchingStatus(String
214
            inputStr, String patternStr){
215
            return !(isNullOrEmpty(inputStr) isNullOrEmpty(
                patternStr));
216
        }
        private static boolean isNullOrEmpty(String str){
218
219
            return ((str == null) (str.isEmpty()));
220
222 }
```