

Mutations of StringUtils

```
1  import java.util.ArrayList;

3  public class StringUtils {

5      public static void main(String[] args){

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);

27         if(Character.compare(escape, '\\') == 0){
28             throw new RuntimeException();
29         }
```

```

31         if(replacement == null){
32             replacement = "";
33         }

35     int charIndex = 0;
    1 Δ int charIndex = 1;

37     boolean underEscapeMode = false;
38     boolean erased;
39     boolean delimiterMode= StringUtils.
        getDelimiterMode(delimiter, inside);
40     while (charIndex < sbPattern.length()){
41     2 Δ while (charIndex <= sbPattern.length()){
42     3 Δ while (charIndex > sbPattern.length()){
43         if(underEscapeMode){
44             underEscapeMode = false;
45             charIndex++;
46         }
47         else{
48             erased = false;
49             if(Character.compare(sbPattern.charAt(
                charIndex), StringUtils.getEscape())
                == 0){
50                 underEscapeMode = true;
51                 sbPattern.deleteCharAt(charIndex);
52                 erased = true;
53             }
54             if(delimiterMode && (Character.compare(
                sbPattern.charAt(charIndex), delimiter
                ) == 0) && !underEscapeMode){
55     4 Δ if(delimiterMode || (Character.compare(
                sbPattern.charAt(charIndex),
                delimiter) == 0) && !underEscapeMode){
56                 sbPattern.deleteCharAt(charIndex);
57                 erased = true;
58             }
59             if(!erased){
60                 charIndex++;
61             }
62         }
63     }

65     if(sbInput.length() < sbPattern.length()){
66         return sbInput.toString();
67     }

```

```

69         if (sbInput.length() == sbPattern.length()) {
70             if (sbInput.toString().equals(sbPattern.
71                 toString()) && !inside) {
72                 return replacement;
73             }
74             else {
75                 return sbInput.toString();
76             }
77         }
78     if (delimiterMode) {
79         ArrayList<Integer> startingPoints = new
80             ArrayList<>();
81         ArrayList<Integer> endingPoints = new
82             ArrayList<>();
83         boolean start = true;
84         for (int i = 0; i < sbInput.length(); i++) {
85             Character currentChar = sbInput.charAt(i)
86                 ;
87             if (Character.compare(delimiter,
88                 currentChar) == 0) {
89                 if (start) {
90                     startingPoints.add(i);
91                     start = false;
92                 }
93                 else {
94                     endingPoints.add(i);
95                     start = true;
96                 }
97             }
98         }
99     if (endingPoints.isEmpty()) {
100         if (inside) {
101             return sbInput.toString();
102         }
103         else {
104             StringUtils.doMatch(sbInput,
105                 sbPattern, replacement, 0, sbInput
106                     .length());
107             return sbInput.toString();
108         }
109     }
110     else {
111         if (startingPoints.get(startingPoints.size
112             () - 1) > endingPoints.get(endingPoints.

```

```

106         size()-1)) {
            startingPoints.remove(startingPoints.
                size()-1);
107     }
108     boolean replaceDone = false;
109     int oldLen;
110     if(inside){
111         for (int i=0; i<startingPoints.size()
112             ; i++){
113             if(startingPoints.get(i)+1 <
114                 endingPoints.get(i)){
115                 oldLen = sbInput.length();
116                 if(doMatch(sbInput, sbPattern
117                     , replacement,
118                         startingPoints.get(i)+1,
119                         endingPoints.get(i))){
120                     replaceDone = true;
121                     updatePoints(
122                         startingPoints,
123                         endingPoints,
124                         startingPoints.get(i),
125                         sbInput.length() -
126                         oldLen);
127                 }
128             }
129         }
130     }
131     if(!replaceDone && (startingPoints.
132         get(0)+1 < endingPoints.get(
133             endingPoints.size()-1))){
134         doMatch(sbInput, sbPattern,
135             replacement, startingPoints.
136                 get(0)+1, endingPoints.get(
137                     endingPoints.size()-1));
138     }
139 }
140 else{
141     int startIndex;
142     int endIndex;
143     if(startingPoints.get(0) > 0){
144         startIndex = 0;
145         oldLen = sbInput.length();
146         if(doMatch(sbInput, sbPattern,
147             replacement, 0, startingPoints.
148                 get(0)) {
149             replaceDone = true;
150             updatePoints(startingPoints,

```

```

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

```

```

158         }
159     }
160     return sbInput.toString();
161 }
162 }
163 else{
164     StringUtils.doMatch(sbInput, sbPattern,
165         replacement, 0, sbInput.length());
166     return sbInput.toString();
167 }
168 }
169
170 private static boolean doMatch(StringBuilder input,
171     StringBuilder pattern, String replace, Integer
172     start, Integer end){
173     String sub = input.substring(start, end);
174     String newSub = StringUtils.replaceAll(sub,
175         pattern.toString(), replace);
176     if(sub.equals(newSub)){
177         return false;
178     }
179     else{
180         input.replace(start, end, newSub);
181         return true;
182     }
183 }
184
185 private static String replaceAll(String source,
186     String from, String to){
187     StringBuilder builder = new StringBuilder(source);
188     ;
189     int index = builder.indexOf(from);
190     while (index != -1)
191     {
192         builder.replace(index, index + from.length(),
193             to);
194         index += to.length();
195         index = builder.indexOf(from, index);
196     }
197     return builder.toString();
198 }
199
200 private static void updatePoints(ArrayList<Integer>
201     startingPoints, ArrayList<Integer> endingPoints,
202     int index, int diff){
203     for(int i=0; i<startingPoints.size(); i++){

```

```

195         if(startingPoints.get(i) > index){
196             startingPoints.set(i, startingPoints.get(
197                 i) + diff);
198         }
199         if(endingPoints.get(i) > index){
200             endingPoints.set(i, endingPoints.get(i) +
201                 diff);
202         }
203     }
204     private static Boolean getDelimiterMode(Character
205         delimiterChar, Boolean insideFlag) throws
206         RuntimeException{
207         if(delimiterChar == null){
208             if(insideFlag){
209                 throw new RuntimeException();
210             }
211             return false;
212         }
213         return true;
214     }
215     private static boolean getMatchingStatus(String
216         inputStr, String patternStr){
217         return !(isEmpty(inputStr) || isEmpty(
218             patternStr));
219     }
220     private static boolean isEmpty(String str){
221         return ((str == null) || (str.isEmpty()));
222     }

```