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

62          if(sbInput.length() < sbPattern.length()){
63              return sbInput.toString();
64          }
```

```
66          if(sbInput.length() == sbPattern.length()){
67              if(sbInput.toString().equals(sbPattern.
                    toString()) && !inside){
68                  return replacement;
69              }
70              else{
71                  return sbInput.toString();
72              }
73          }

75          if(delimiterMode){
76              ArrayList<Integer> startingPoints = new
                    ArrayList<>();
77              ArrayList<Integer> endingPoints = new
                    ArrayList<>();
78              boolean start = true;
79              for (int i = 0; i < sbInput.length(); i++){
                5Δ  for (int i = 1; i < sbInput.length(); i
                    ++){
80                  Character currentChar = sbInput.charAt(i)
                        ;
81                  if(Character.compare(delimiter,
                        currentChar) == 0){
82                      if(start) {
83                          startingPoints.add(i);
84                          start = false;
85                      }
86                      else{
87                          endingPoints.add(i);
88                          start = true;
                            6Δ  // start = true;
89                      }
90                  }
91              }
92              if(endingPoints.isEmpty()){
93                  if(inside){
94                      return sbInput.toString();
95                  }
96                  else{
97                      StringUtils.doMatch(sbInput,
                            sbPattern, replacement, 0, sbInput
                            .length());
98                      return sbInput.toString();
99                  }
100             }
101             else{
```

```
102                 if(startingPoints.get(startingPoints.size
                        ()-1) > endingPoints.get(endingPoints.
                        size()-1)){
103                     startingPoints.remove(startingPoints.
                            size()-1);
104                 }
105                 boolean replaceDone = false;
106                 int oldLen;
107                 if(inside){
108                     for (int i=0; i<startingPoints.size()
                            ; i++){
109                         if(startingPoints.get(i)+1 <
                                endingPoints.get(i)){
                    4Δ      if(startingPoints.get(i)+1 <=
                                endingPoints.get(i)){
110                             oldLen = sbInput.length();
111                             if(doMatch(sbInput, sbPattern
                                    ,replacement,
                                    startingPoints.get(i)+1,
                                    endingPoints.get(i))){
112                                 replaceDone = true;
113                                 updatePoints(
                                        startingPoints,
                                        endingPoints,
                                        startingPoints.get(i),
                                         sbInput.length() -
                                        oldLen);
114                             }
115                         }
116                     }
117                     if(!replaceDone && (startingPoints.
                            get(0)+1 < endingPoints.get(
                            endingPoints.size()-1))){
118                         doMatch(sbInput, sbPattern,
                                replacement, startingPoints.
                                get(0)+1, endingPoints.get(
                                endingPoints.size()-1));
119                     }
120                 }
121                 else{
122                     int startIndex;
123                     int endIndex;
124                     if(startingPoints.get(0) > 0){
125                         startIndex = 0;
126                         oldLen = sbInput.length();
127                         if(doMatch(sbInput, sbPattern,
```

```
                    replacement,0, startingPoints.
                    get(0))) {
128                  replaceDone = true;
129                  updatePoints(startingPoints,
                         endingPoints, 0, sbInput.
                         length() - oldLen);
130              }
131          }
132          else{
133              startIndex = endingPoints.get(0)
                     +1;
134          }
135          if(endingPoints.get(endingPoints.size
                 ()-1)+1 < sbInput.length()){
136              endIndex = sbInput.length();
137              if(doMatch(sbInput, sbPattern,
                     replacement, endingPoints.get(
                     endingPoints.size()-1)+1,
                     sbInput.length())) {
138                  replaceDone = true;
139              }
140          }
141          else{
142              endIndex = startingPoints.get(
                     startingPoints.size()-1) -1 ;
143          }
144          for(int i=0; i<endingPoints.size()-1;
                 i++){
145              if(endingPoints.get(i)+1 <
                     startingPoints.get(i+1)){
146                  oldLen = sbInput.length();
147                  if(doMatch(sbInput, sbPattern
                         ,replacement,endingPoints.
                         get(i)+1, startingPoints.
                         get(i+1))){
148                      replaceDone = true;
149                      updatePoints(
                             startingPoints,
                             endingPoints,
                             endingPoints.get(i),
                             sbInput.length() -
                             oldLen);
150                  }
151              }
152          }
153          if(!replaceDone && (startIndex <
```

```
                        endIndex)){
154                      doMatch(sbInput, sbPattern,
                             replacement, startIndex,
                             endIndex);
155                 }
156             }
157             return  sbInput.toString();
158         }
159     }
160     else{
161         StringUtils.doMatch(sbInput, sbPattern,
                replacement, 0, sbInput.length());
162         return sbInput.toString();
163     }
164 }

166 private static boolean doMatch(StringBuilder input,
        StringBuilder pattern, String replace, Integer
        start, Integer end){
167     String sub = input.substring(start, end);
168     String newSub = StringUtils.replaceAll(sub,
            pattern.toString(), replace);
169     if(sub.equals(newSub)){
170         return false;
171     }
172     else{
173         input.replace(start, end, newSub);
174         return true;
175     }
176 }

178 private static String replaceAll(String source,
        String from, String to){
179     StringBuilder builder = new StringBuilder(source)
            ;
180     int index = builder.indexOf(from);
181     while (index != -1)
182     {
183         builder.replace(index, index + from.length(),
                to);
184         index += to.length();
185         index = builder.indexOf(from, index);
186     }
187     return builder.toString();
188 }
```

```
190     private static void updatePoints(ArrayList<Integer>
            startingPoints, ArrayList<Integer> endingPoints,
            int index, int diff){
191         for(int i=0; i<startingPoints.size(); i++){
192             if(startingPoints.get(i) > index){
193                 startingPoints.set(i, startingPoints.get(
                        i) + diff);
194             }
195             if(endingPoints.get(i) > index){
196                 endingPoints.set(i, endingPoints.get(i) +
                        diff);
197             }
198         }
199     }

201     private static Boolean getDelimiterMode(Character
            delimiterChar, Boolean insideFlag) throws
            RuntimeException{
202         if(delimiterChar == null){
203             if(insideFlag){
204                 throw new RuntimeException();
205             }
206             return false;
207         }
208         return true;
209     }

211     private static boolean getMatchingStatus(String
            inputStr, String patternStr){
212         return !(isNullOrEmpty(inputStr)  isNullOrEmpty(
                patternStr));
213     }

215     private static boolean isNullOrEmpty(String str){
216         return ((str == null)  (str.isEmpty()));
217     }

219 }
```