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                     4Δ  for (int i=0; i<startingPoints.
                            size(); i+2){
110                         if(startingPoints.get(i)+1 <
                                endingPoints.get(i)){
111                             oldLen = sbInput.length();
112                             if(doMatch(sbInput, sbPattern
                                    ,replacement,
                                    startingPoints.get(i)+1,
                                    endingPoints.get(i))){
113                                 replaceDone = true;
114                                 updatePoints(
                                        startingPoints,
                                        endingPoints,
                                        startingPoints.get(i),
                                         sbInput.length() -
                                        oldLen);
115                             }
116                         }
117                     }
118                     if(!replaceDone && (startingPoints.
                            get(0)+1 < endingPoints.get(
                            endingPoints.size()-1))){
119                         doMatch(sbInput, sbPattern,
                                replacement, startingPoints.
                                get(0)+1, endingPoints.get(
                                endingPoints.size()-1));
120                     }
121                 }
122                 else{
123                     int startIndex;
124                     int endIndex;
125                     if(startingPoints.get(0) > 0){
126                         startIndex = 0;
127                         oldLen = sbInput.length();
128                         if(doMatch(sbInput, sbPattern,
```

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

```
                        endIndex)){
155                      doMatch(sbInput, sbPattern,
                             replacement, startIndex,
                             endIndex);
156                 }
157             }
158             return  sbInput.toString();
159         }
160     }
161     else{
162         StringUtils.doMatch(sbInput, sbPattern,
                replacement, 0, sbInput.length());
163         return sbInput.toString();
164     }
165 }

167 private static boolean doMatch(StringBuilder input,
        StringBuilder pattern, String replace, Integer
        start, Integer end){
168     String sub = input.substring(start, end);
169     String newSub = StringUtils.replaceAll(sub,
            pattern.toString(), replace);
170     if(sub.equals(newSub)){
171         return false;
172     }
173     else{
174         input.replace(start, end, newSub);
175         return true;
176     }
177 }

179 private static String replaceAll(String source,
        String from, String to){
180     StringBuilder builder = new StringBuilder(source)
            ;
181     int index = builder.indexOf(from);
182     while (index != -1)
183     {
184         builder.replace(index, index + from.length(),
                to);
185         index += to.length();
186         index = builder.indexOf(from, index);
187     }
188     return builder.toString();
189 }
```

```
191     private static void updatePoints(ArrayList<Integer>
            startingPoints, ArrayList<Integer> endingPoints,
            int index, int diff){
192         for(int i=0; i<startingPoints.size(); i++){
193             if(startingPoints.get(i) > index){
194                 startingPoints.set(i, startingPoints.get(
                        i) + diff);
195             }
196             if(endingPoints.get(i) > index){
197                 endingPoints.set(i, endingPoints.get(i) +
                        diff);
198             }
199         }
200     }

202     private static Boolean getDelimiterMode(Character
            delimiterChar, Boolean insideFlag) throws
            RuntimeException{
203         if(delimiterChar == null){
204             if(insideFlag){
205                 throw new RuntimeException();
206             }
207             return false;
208         }
209         return true;
210     }

212     private static boolean getMatchingStatus(String
            inputStr, String patternStr){
213         return !(isNullOrEmpty(inputStr)  isNullOrEmpty(
                patternStr));
214     }

216     private static boolean isNullOrEmpty(String str){
217         return ((str == null)  (str.isEmpty()));
218     }

220 }
```