Mutations of StringUtils

```java
import java.util.ArrayList;

public class StringUtils {

    public static void main(String[] args){


    }

    private static volatile char escape = 'e';

    public static char getEscape() {
        return escape;
    }

    public static void setEscape(char escape) {
        StringUtils.escape = escape;
    }

    public static String replaceString(String inputText, String pattern, String replacement, Character delimiter, boolean inside) throws RuntimeException {
        if(!StringUtils.getMatchingStatus(inputText, pattern)){
            return inputText;
        }
        StringBuilder sbInput = new StringBuilder(inputText);
        StringBuilder sbPattern = new StringBuilder(pattern);

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

```
if(replacement == null){
    replacement = "";
}

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


boolean underEscapeMode = false;
boolean erased;
boolean delimiterMode= StringUtils.
    getDelimiterMode(delimiter, inside);
while (charIndex < sbPattern.length()){
2 Δ  while (charIndex <= sbPattern.length()){
3 Δ  while (charIndex > sbPattern.length()){
    if(underEscapeMode){
        underEscapeMode = false;
        charIndex++;
    }
    else{
        erased = false;
        if(Character.compare(sbPattern.charAt(
            charIndex), StringUtils.getEscape())
            == 0){
            underEscapeMode = true;
            sbPattern.deleteCharAt(charIndex);
            erased = true;
        }
        if(delimiterMode && (Character.compare(
            sbPattern.charAt(charIndex), delimiter
            ) == 0) && !underEscapeMode){
        4 Δ  if(delimiterMode || (Character.compare
            (sbPattern.charAt(charIndex),
            delimiter) == 0) && !underEscapeMode){
            sbPattern.deleteCharAt(charIndex);
            erased = true;
        }
        if(!erased){
            charIndex++;
        }
    }
}

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

```java
        if(sbInput.length() == sbPattern.length()){
            if(sbInput.toString().equals(sbPattern.toString()) && !inside){
                return replacement;
            }
            else{
                return sbInput.toString();
            }
        }

        if(delimiterMode){
            ArrayList<Integer> startingPoints = new ArrayList<>();
            ArrayList<Integer> endingPoints = new ArrayList<>();
            boolean start = true;
            for (int i = 0; i < sbInput.length(); i++){
                Character currentChar = sbInput.charAt(i);
                if(Character.compare(delimiter, currentChar) == 0){
                    if(start) {
                        startingPoints.add(i);
                        start = false;
                    }
                    else{
                        endingPoints.add(i);
                        start = true;
                    }
                }
            }
            if(endingPoints.isEmpty()){
                if(inside){
                    return sbInput.toString();
                }
                else{
                    StringUtils.doMatch(sbInput, sbPattern, replacement, 0, sbInput.length());
                    return sbInput.toString();
                }
            }
            else{
                if(startingPoints.get(startingPoints.size()-1) > endingPoints.get(endingPoints.
                    size()-1)){
                    startingPoints.remove(startingPoints.size()-1);
                }
                boolean replaceDone = false;
                int oldLen;
                if(inside){
                    for (int i=0; i<startingPoints.size(); i++){
                        if(startingPoints.get(i)+1 < endingPoints.get(i)){
                            oldLen = sbInput.length();
                            if(doMatch(sbInput, sbPattern,replacement, startingPoints.get(i)+1, endingPoints.get(i))){
                                replaceDone = true;
                                updatePoints(startingPoints, endingPoints, startingPoints.get(i), sbInput.length() - oldLen);
                            }
                        }
                    }
                    if(!replaceDone && (startingPoints.get(0)+1 < endingPoints.get(endingPoints.size()-1))){
                        doMatch(sbInput, sbPattern, replacement, startingPoints.get(0)+1, endingPoints.get(endingPoints.size()-1));
                    }
                }
                else{
                    int startIndex;
                    int endIndex;
                    if(startingPoints.get(0) > 0){
                        startIndex = 0;
                        oldLen = sbInput.length();
                        if(doMatch(sbInput, sbPattern, replacement,0, startingPoints.get(0))) {
                            replaceDone = true;
                            updatePoints(startingPoints,
                                endingPoints, 0, sbInput.length() - oldLen);
                        }
                    }
                    else{
                        startIndex = endingPoints.get(0)+1;
                    }
                    if(endingPoints.get(endingPoints.size()-1)+1 < sbInput.length()){
                        endIndex = sbInput.length();
                        if(doMatch(sbInput, sbPattern, replacement, endingPoints.get(endingPoints.size()-1)+1, sbInput.length())) {
                            replaceDone = true;
                        }
                    }
                    else{
                        endIndex = startingPoints.get(startingPoints.size()-1) -1 ;
                    }
                    for(int i=0; i<endingPoints.size()-1; i++){
                        if(endingPoints.get(i)+1 < startingPoints.get(i+1)){
                            oldLen = sbInput.length();
                            if(doMatch(sbInput, sbPattern,replacement,endingPoints.get(i)+1, startingPoints.get(i+1))){
                                replaceDone = true;
                                updatePoints(startingPoints, endingPoints, endingPoints.get(i), sbInput.length() - oldLen);
                            }
                        }
                    }
                    if(!replaceDone && (startIndex < endIndex)){
                        doMatch(sbInput, sbPattern, replacement, startIndex, endIndex);
```

```
                    }
                }
                return  sbInput.toString();
            }
        }
        else{
            StringUtils.doMatch(sbInput, sbPattern, replacement, 0, sbInput.length());
            return sbInput.toString();
        }
    }

    private static boolean doMatch(StringBuilder input, StringBuilder pattern, String replace, Integer start, Integer end){
        String sub = input.substring(start, end);
        String newSub = StringUtils.replaceAll(sub, pattern.toString(), replace);
        if(sub.equals(newSub)){
            return false;
        }
        else{
            input.replace(start, end, newSub);
            return true;
        }
    }

    private static String replaceAll(String source, String from, String to){
        StringBuilder builder = new StringBuilder(source);
        int index = builder.indexOf(from);
        while (index != -1)
        {
            builder.replace(index, index + from.length(), to);
            index += to.length();
            index = builder.indexOf(from, index);
        }
        return builder.toString();
    }

    private static void updatePoints(ArrayList<Integer> startingPoints, ArrayList<Integer> endingPoints, int index, int diff){
        for(int i=0; i<startingPoints.size(); i++){
```

```java
            if(startingPoints.get(i) > index){
                startingPoints.set(i, startingPoints.get(i) + diff);
            }
            if(endingPoints.get(i) > index){
                endingPoints.set(i, endingPoints.get(i) + diff);
            }
        }
    }

    private static Boolean getDelimiterMode(Character delimiterChar, Boolean insideFlag) throws RuntimeException{
        if(delimiterChar == null){
            if(insideFlag){
                throw new RuntimeException();
            }
            return false;
        }
        return true;
    }

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

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

}
```