简单的解压缩算法

知识点栈
时间限制：1s 空间限制：256MB 限定语言：不限

题目描述：

现需要实现一种算法，能将一组压缩字符串还原成原始字符串，还原规则如下：
1、字符后面加数字N，表示重复字符N次。例如：压缩内容为A3，表示原始字符串为AAA。
2、花括号中的字符串加数字N，表示花括号中的字符串重复N次。例如：压缩内容为{AB}3，表示原始字符串为ABABAB。
3、字符加N和花括号后面加N，支持任意的嵌套，包括互相嵌套。例如：压缩内容可以{A3B1{C}3}3。

输入描述：

输入一行压缩后的字符串

输出描述：

输出压缩前的字符串

补充说明：

输入保证，数字不会为0，花括号中的内容不会为空，保证输入的都是合法有效的压缩字符串
输入输出字符串区分大小写
输入的字符串长度为范围[1, 10000]
输出的字符串长度为范围[1, 100000]
数字N范围[1, 10000]

示例1

输入：

A3

输出：

AAA

说明：

A3代表A字符重复3次

示例2

输入：

{A3B1{C}3}3

输出：

AAABCCCAAABCCCAAABCCC

说明：

{A3B1{C}3}3代表A字符重复3次，B字符重复1次，花括号中的C字符重复3次，最外层花括号中的AAABCCC重复3次

解题思路：

例如：{A3B1{C}3{x2Y}2L4}3
使用双向队列deque来放置字符串，使用tempStr来放置临时字符串，isHasBrace代表是有大括号需要处理，初始化false
1、从头开始遍历，遍历到“{”，isHasBrace=false且tempStr为空，则deque=【{】；
继续遍历，直到第二个“{”，isHasBrace=false但是tempStr有值，则进行处理后放入队列中，deque=【{，AAAB，{】；
2、遍历到“}”，tempStr=“C”，因为isHasBrace=false，所以先处理完tempStr放入队列中，deque=【{，AAAB，{，C】，isHasBrace置为true；
3、遍历到“{”，因为isHasBrace=true，需要处理大括号：遍历队列，发现括号里面只有一个C，此时tempStr=3，处理完放入队列中：deque=【{，AAAB，CCC，{】，isHasBrace置为false；
4、遍历到“}”，tempStr=“x2Y”，因为isHasBrace=false，所以先处理完tempStr放入队列中，deque=【{，AAAB，CCC，{，xxY】，isHasBrace置为true；
5、遍历到“L”，因为L是字符且isHasBrace=true，需要处理大括号：遍历队列，发现括号里面是xxY，此时tempStr=2，处理完放入队列中：deque=【{，AAAB，CCC，xxYxxY】，isHasBrace置为false；
6、遍历到“}”，tempStr=“L4”，因为isHasBrace=false，所以先处理完tempStr放入队列中，deque=【{，AAAB，CCC，xxYxxY，LLLL】，isHasBrace置为true；
7、遍历到最后tempStr=3，isHasBrace=true，需要处理大括号：遍历队列，发现括号里面是AAABCCCxxYxxYLLLL，此时tempStr=3，处理完放入队列中：deque=【AAABCCCxxYxxYLLLLAAABCCCxxYxxYLLLLAAABCCCxxYxxYLLLL】。
最终从头输出队列为：AAABCCCxxYxxYLLLLAAABCCCxxYxxYLLLLAAABCCCxxYxxYLLLL

```
public class Main{

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        String string = sc.nextLine();

        int len = string.length();
        Deque<String> strDeque = new ArrayDeque<>();
        String tempStr = "";       //字符串
        boolean isHasBraces = false;       //是否有大括号需要处理
        for(int i=0; i<len; i++){
            char c = string.charAt(i);
            if(c == '{'){            //遇到左括号
                if(isHasBraces){       //先判断是否有大括号需要处理
                    handleBraces( tempStr, strDeque);
                    isHasBraces = false;
                }else if(!tempStr.isEmpty()){
```

```
                    strDeque.addLast(handle(tempStr));
                }
                strDeque.addLast(String.valueOf(c));
                tempStr = "";
            }else if(c == '}'){
                strDeque.addLast(tempStr);
                if(isHasBraces){
                    handleBraces( strDeque.pollLast(), strDeque);
                }else {
                    strDeque.addLast(handle(strDeque.pollLast()));
                }
                isHasBraces = true;
                tempStr = "";
            }else {
                if(isHasBraces && Character.isLetter(c)){
                    handleBraces( tempStr, strDeque);
                    isHasBraces = false;
                    tempStr = "";
                }
                tempStr += c;
            }
            if(i == len - 1){     //最后一个字符
                if(isHasBraces){
                    handleBraces( tempStr, strDeque);
                }else {
                    strDeque.addLast(handle(tempStr));
                }
            }
        }

        String res = "";
        while (strDeque.size() != 0){
            res += strDeque.pollFirst();
        }

        System.out.println(res);
    }

    /**
     * 获取大括号里面的字符串
     * @param strDeque
     * @return
     */
    public static String getBraces(Deque<String> strDeque){
```

```
        List<String> list = new ArrayList<>();
        while (!strDeque.peekLast().equals("{")){     //直至碰到下一个左括号
            list.add(strDeque.pollLast());
        }
        strDeque.pollLast();       //左括号也需要删除
        Collections.reverse(list);    //因为是从尾部开始找的所以需要翻转

        String res = "";
        for(String s : list){
            res += s;
        }

        return res;
    }

    /**
     * 处理大括号里面的内容
     * @param tempStr           括号外的数字
     * @param strDeque          字符串队列
     */
    public static void handleBraces(String tempStr, Deque<String> strDeque){

        int num = Integer.valueOf(tempStr);
        String temp = "";
        String strInBraces = getBraces(strDeque);
        for(int j=0; j<num; j++){
            temp += strInBraces;
        }
        strDeque.addLast(temp);
    }

    /**
     * 处理连续的字符串
     * @param str
     * @return
     */
    public static String handle(String str){

        int len = str.length();

        String res = "";
        String tempStr = "";       //字符串
        String tempNum = "";       //数字字符串（处理多位数）
```

```
        for(int i=0; i<len; i++){
            char c = str.charAt(i);
            if(Character.isDigit(c)){     //此时是数字
                tempNum += c;     //是数字直接进行拼接
            }else {         //此时是字母
                if(!tempNum.isEmpty()){         //数字字符串不为空说明需要处理
                    int num = Integer.valueOf(tempNum);         //字符重复的次数
                    for(int j=0; j<num; j++){
                        res += tempStr;         //重复多少次就拼接多少次
                    }
                    tempStr = "";     //处理完需要置空以免影响下一个字符的统计
                    tempNum = "";     //数字一样置空
                }
                tempStr += c;     //字符拼接
            }
            if(i == len - 1){     //不能忘了处理最后一位
                if(!tempNum.isEmpty()){         //数字字符串不为空时处理
                    int num = Integer.valueOf(tempNum);
                    for(int j=0; j<num; j++){
                        res += tempStr;
                    }
                }else {
                    res += tempStr;         //为空时直接拼接字符串（处理单字符串的情况）
                }
            }
        }

        return res;
    }

}
```