Run length encoding and decoding

Write code which encodes a string using run-length encoding and decodes a string encoded using run-length encoding

Example:

"abbccc" will be encoded as "1a2b3c",
"aabbbcccc" will be encoded as "2a3b4c"
"1d2e1f" will be decoded as "deef"

Assume only letters are present in the string to be encoded, no numerals

Remember to handle the case where we can have >9 of the same characters in a row

RUN LENGTH ENCODING - ENCODE FUNCTION

```java
public static String encode(String originalString) {
    if (originalString == null) {
        return null;
    }

    StringBuilder sb = new StringBuilder();
    int currIndex = 0;
    while (currIndex < originalString.length()) {
        char currChar = originalString.charAt(currIndex);

        int num = 0;
        int compareIndex = currIndex;
        while (compareIndex < originalString.length()
                && currChar == originalString.charAt(compareIndex)) {
            compareIndex++;
            num++;
        }
        sb.append(num);
        sb.append(currChar);

        currIndex = compareIndex;
    }

    return sb.toString();
}
```

A NULL STRING IS ENCODED AS NULL

START AT INDEX 0 AND READ THE STRING CHARACTER BY CHARACTER

WALK THE STRING, ENSURING THAT YOU STAY WITHIN BOUNDS

COMPARE THE CURRENT CHARACTER TO ALL CHARACTERS FOLLOWING IT TO SEE IF IT IS THE SAME

"NUM" KEEPS TRACK OF HOW MANY TIMES THE CHARACTER IS REPEATED

APPEND THE NUMBER OF TIMES THE CHARACTER APPEARS IN ONE "RUN" AND THE CHARACTER ITSELF

MOVE TO THE NEXT CHARACTER AFTER THE CURRENT "RUN"

CONSIDER THE CASES TO HANDLE WHILE DECODING

1a22b1c

A NUMBER IS FOLLOWED BY THE CHARACTER, THE NUMBER REPRESENTS HOW MANY TIMES THE CHARACTER IS REPEATED

THE NUMBER CAN SPAN MULTIPLE CHARACTERS IF IT IS >9

WHEN READING A NUMBER THE CODE SHOULD SCAN THE STRING TILL IT FINDS THE END OF THE NUMBER, THEN GET ITS NUMERIC VALUE

RUN LENGTH ENCODING - DECODE FUNCTION

```java
public static String decode(String encodedString) {
    if (encodedString == null) {
        return null;
    }

    StringBuilder sb = new StringBuilder();
    int numStartIndex = 0;
    int numEndIndex = 1;
    while (numEndIndex < encodedString.length()) {
        while (Character.isDigit(encodedString.charAt(numEndIndex))) {
            numEndIndex++;
        }

        int charIndex = numEndIndex;
        String numString = encodedString.substring(numStartIndex, numEndIndex);
        int num = Integer.valueOf(numString);
        for (int i = 0; i < num; i++) {
            sb.append(encodedString.charAt(charIndex));
        }
        numStartIndex = charIndex + 1;
        numEndIndex = numStartIndex + 1;
    }

    return sb.toString();
}
```

A NULL STRING IS DECODED AS NULL

THE ENCODED STRING WILL HAVE A NUMBER FOLLOWED BY THE CHARACTER WHICH IS REPEATED THAT NUMBER OF TIMES.

THE DIGITS OF THE NUMBER ARE CHARACTERS IN THE STRING AND THESE INDICES KEEP TRACK OF HOW MANY CHARACTERS THE NUMBER OCCUPIES, THE NUMBER CAN BE >9 WHICH MEANS IT OCCUPIES MORE THAN ONE CHARACTER POSITION

THE CHARACTER IS PRESENT AT THIS POSITION, AFTER THE NUMBER DENOTING HOW OFTEN THE CHARACTER IS REPEATED

APPEND THE CHARACTER TO THE DECODED STRING, AS MANY TIMES AS THE ENCODING SPECIFIED

MOVE TO THE NEXT NUMBER + CHARACTER WHICH HAS TO BE DECODED

Add two numbers represented by their digits

Given two numbers where the individual digits in the numbers are in an array or a list add them to get the final result in the same list or array form

Example using arrays: {1, 2} represents the number 12. Note that the most significant digit in the 0th index of the array, with the least significant digit at the last position.

Adding {1,2} and {2,3} should give the result {3,5}

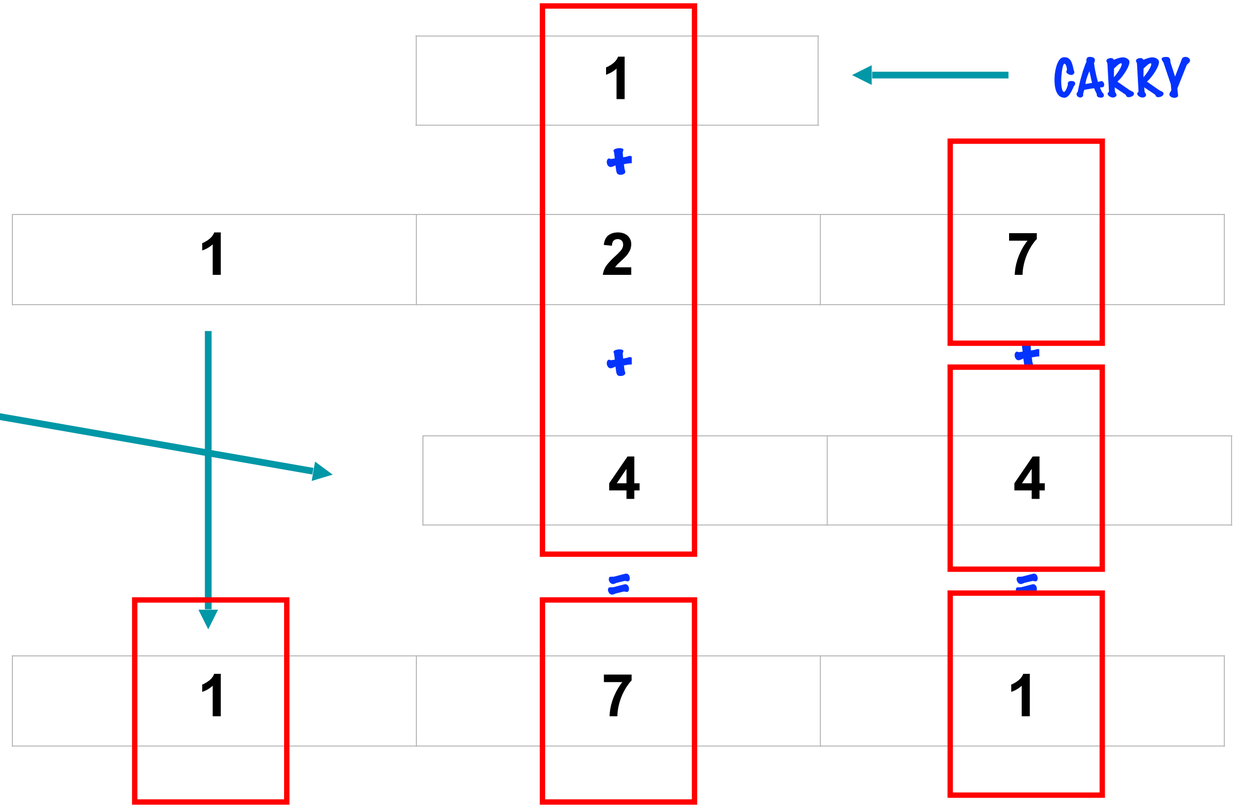
Don't convert the number format to a real number to add them, add them digit by digit.

HINT: Remember to consider the carry over per digit if there is one!

ADD TWO NUMBERS REPRESENTED BY THEIR DIGITS

SET UP THE NUMBERS IN A LIST OR AN ARRAY, DIGIT BY DIGIT

ADD EACH INDIVIDUAL DIGIT AND STORE THE RESULT IN ANOTHER LIST OR ARRAY



START WITH THE LEAST SIGNIFICANT OR RIGHTMOST DIGITS

NOTE, IF THE SUM OF ANY 2 INDIVIDUAL DIGITS CROSS 10 THEN WE HAVE A "CARRY" WHICH GOES OVER TO THE NEXT DIGIT ON THE LEFT

ADD TWO NUMBERS REPRESENTED BY THEIR DIGITS

```java
public static int[] addNumbers(int[] num1, int[] num2) {
    List<Integer> digitList = new ArrayList<~>();
    int lastIndex1 = num1.length - 1;
    int lastIndex2 = num2.length - 1;

    int carry = 0;
    int total = 0;
    int digit = 0;
    while (lastIndex1 >= 0 && lastIndex2 >= 0) {
        total = num1[lastIndex1] + num2[lastIndex2] + carry;
        digit = total % 10;
        carry = total / 10;

        digitList.add(0, digit);
        lastIndex1--;
        lastIndex2--;
    }
    while (lastIndex1 >= 0) {
        total = num1[lastIndex1] + carry;
        digit = total % 10;
        carry = total / 10;

        digitList.add(0, digit);
        lastIndex1--;
    }
    while (lastIndex2 >= 0) {
        total = num2[lastIndex2] + carry;
        digit = total % 10;
        carry = total / 10;

        digitList.add(0, digit);
        lastIndex2--;
    }
    if (carry != 0) {
        digitList.add(0, carry);
    }

    int[] sum = new int[digitList.size()];
    for (int i = 0; i < digitList.size(); i++) {
        sum[i] = digitList.get(i);
    }

    return sum;
}
```

STORE THE LENGTHS OF THE TWO NUMBERS

THE INITIAL CARRY IS 0, WHEN WE START THE ADDITION FROM THE LEAST SIGNIFICANT DIGIT

IF THE TOTAL WAS 17 THEN 17 % 10 IS THE MODULO OPERATOR WILL GIVE US 7. THIS IS THE DIGIT THAT WILL BE STORED AS THE RESULT

IF THE TOTAL > 10 THEN CARRY WILL BE 1 OTHERWISE 0

IF ONE NUMBER IS LARGER WITH MORE DIGITS, ENSURE THAT THOSE DIGITS ARE ALSO ADDED TO GET THE SUM

NOTE THAT THE LAST DIGITS CAN ALSO GENERATE A CARRY!

CONVERT THE NUMBER STORED IN A LIST TO AN ARRAY