Assignment 7 Solution (String)

Question 1:Given two strings s and t, *determine if they are isomorphic*.

Two strings s and t are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.

Solution:

```
 public boolean isIsomorphic(String s, String t) {


        int map1[]=new int[200];

        int map2[]=new int[200];


        if(s.length()!=t.length())

            return false;




        for(int i=0;i<s.length();i++)

        {

            if(map1[s.charAt(i)]!=map2[t.charAt(i)])

                return false;


            map1[s.charAt(i)]=i+1;
```

```
            map2[t.charAt(i)]=i+1;
        }
        return true;
    }
```

Question 2

Given a string num which represents an integer, return true *if* num *is a strobogrammatic number*.

A strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).

Solution:

```
 public boolean isStrobogrammatic(String num) {
        Map<Character, Character> map = new HashMap<Character, Character>();
        map.put('6', '9');
        map.put('9', '6');
        map.put('0', '0');
        map.put('1', '1');
        map.put('8', '8');

        int l = 0, r = num.length() - 1;
        while (l <= r) {
            if (!map.containsKey(num.charAt(l))) return false;
            if (map.get(num.charAt(l)) != num.charAt(r))
                return false;
```

```
            l++;
            r--;
        }

        return true;
    }
```

Question 3

Given two non-negative integers, num1 and num2 represented as string, return *the sum of* num1 *and* num2 *as a string*.

You must solve the problem without using any built-in library for handling large integers (such as BigInteger). You must also not convert the inputs to integers directly.

Solution:

```
public String addStrings(String num1, String num2) {
        StringBuilder sb = new StringBuilder();

        int i = num1.length() - 1, j = num2.length() - 1;
        int carry = 0;

        while (i >= 0 || j >= 0) {
            int sum = carry;

            if (i >= 0) sum += (num1.charAt(i--) - '0');
```

```
            if (j >= 0) sum += (num2.charAt(j--) - '0');
            sb.append(sum % 10);
            carry = sum / 10;
        }
        if (carry != 0) sb.append(carry);
        return sb.reverse().toString();
    }
```

Question 4

Given a string s, reverse the order of characters in each word within a sentence while still preserving whitespace and initial word order.

Solution:

```
 public String reverseWords(String s) {
        int l=0;
        int r=0;
        char a[]=s.toCharArray();
        while(l<s.length()){
            while(r<s.length() && a[r]!=' '){
                r++;
            }
            //reverse part


            //r-1 because r is pointing current at blank space
```

```
            //Let's
            //    |
            //    r
            reverse(a,l,r-1);
            l=r+1;
            r=l;
        }
        return String.valueOf(a);
    }
     public String reverse(char s[],int l,int r){
         while(l<r){
             char temp=s[l];
             s[l]=s[r];
             s[r]=temp;
             l++;
             r--;
         }
         return String.valueOf(s);
     }
```

Question 5

Given a string s and an integer k, reverse the first k characters for every 2k characters counting from the start of the string.

If there are fewer than k characters left, reverse all of them. If there are less than 2k but greater than or equal to k characters, then reverse the first k characters and leave the other as original.

Solution:

```
 public String reverseStr(String s, int k) {


    int i=0;


    String ans="";


    while(i<s.length()){


    if(i+(2*k)-1<s.length()||(i+(2*k)-1>=s.length()&&s.length()-i>=k)){

    int index=i+(2*k)-1;

    StringBuilder str=new StringBuilder();

    String app=s.substring(i,i+k);

    str.append(app);

    str.reverse();

    ans+=str.toString();

    i=i+k;

    if(index<s.length()){

        ans+=s.substring(i,index+1);
```

```
    }
    else{
        ans+=s.substring(i,s.length());
    }

    i=index+1;


    }


    else{
    StringBuilder temp2=new StringBuilder();
    temp2.append(s.substring(i,s.length()));
    temp2.reverse();
    ans+=temp2.toString();
    break;
    }


    }


    return ans;


    }
```

Question 6

Given two strings s and goal, return true *if and only if* s *can become* goal *after some number of* ***shifts*** *on* s.

A **shift** on s consists of moving the leftmost character of s to the rightmost position.

- For example, if s = "abcde", then it will be "bcdea" after one shift.

Solution:

```
public boolean rotateString(String s, String goal) {
    if(s ==null || goal == null){
        return false;
    }
    if(s.length() != goal.length()) return false;
    if(s.length() == 0){
        return true;
    }
    int i =0, j =0;
    while(i < s.length() && j < goal.length()){
        if(s.charAt(i) == goal.charAt(j)){
            i++; j++;
        }
        else{
            if(j == 0){
                i++;
            }
```

```
            else{
                j= 0;
            }
        }
    }
    return (s.substring(0,goal.length() - j).equals(goal.substring(j)));
}
```

Question 7

Given two strings s and t, return true *if they are equal when both are typed into empty text editors*. '#' means a backspace character.

Note that after backspacing an empty text, the text will continue empty.

Solution:

```
public boolean backspaceCompare(String s, String t) {
    return buildString(s).equals(buildString(t));
}


private String buildString(String str) {
    Stack<Character> stack = new Stack<>();
    for (char c : str.toCharArray()) {
        if (c != '#') {
            stack.push(c);
        } else if (!stack.isEmpty()) {
            stack.pop();
```

```
            }
        }
        StringBuilder sb = new StringBuilder();
        while (!stack.isEmpty()) {
            sb.append(stack.pop());
        }
        return sb.toString();
    }
```

Question 8

You are given an array coordinates, coordinates[i] = [x, y], where [x, y] represents the coordinate of a point. Check if these points make a straight line in the XY plane.

Solution:

```
    public boolean checkStraightLine(int[][] coordinates) {
        int n = coordinates.length;

        int x1 = coordinates[0][0];
        int y1 = coordinates[0][1];

        int x2 = coordinates[1][0];
        int y2 = coordinates[1][1];

        for (int i = 2; i < n; i++) {
```

```
        int x = coordinates[i][0];
        int y = coordinates[i][1];

        if ((y - y1) * (x - x2) != (y - y2) * (x - x1)) {
            return false;
        }
    }

    return true;
}
```