```
public class RecursionStriver {
//*********************************Printing
Subsequences*************************************************************** */
class Solution{
    public static void main(String[] args) {
        int arr[]={3,1,2};
        ArrayList<ArrayList<Integer>> ds=new ArrayList<>();
        ArrayList<Integer> ans=new ArrayList<>();
        printSubsequences(arr,0,3,ds,ans);
        System.out.println(ds);
        }
        public static void printSubsequences(int arr[],int ind,int n,ArrayList<ArrayList<Integer>>
ds,ArrayList<Integer> ans) {
            if(ind==n){
                ds.add(new ArrayList<>(ans));
                return;
            }
            ans.add(arr[ind]);
            printSubsequences(arr,ind+1,n,ds,ans);
            ans.remove(ans.size()-1);
            printSubsequences(arr,ind+1,n,ds,ans);
        }
    }
//*********************************Printing Subsequences whose sum is
K********************************************** */
class Solution{
    public static void main(String[] args) {
    int arr[]={1,2,1};
    int sum=2;
    ArrayList<ArrayList<Integer>> ds=new ArrayList<>();
    ArrayList<Integer> ans=new ArrayList<>();
    PrintSubsequencesWhoseSumisK(arr,0,3,ds,ans,0,sum);
    System.out.println(ds);
    }
    public static void PrintSubsequencesWhoseSumisK(int arr[],int ind,int
n,ArrayList<ArrayList<Integer>> ds,ArrayList<Integer> ans,int s,int sum) {
        //System.out.println("ps"+" "+ind+" "+n+" "+ ans+" "+ds);
        if(ind==n){
            if(s==sum){
            ds.add(new ArrayList<>(ans));
            return;
            }
            else{return;}
        }
        ans.add(arr[ind]);
        s+=arr[ind];
        PrintSubsequencesWhoseSumisK(arr,ind+1,n,ds,ans,s,sum);
        ans.remove(ans.size()-1);
        s-=arr[ind;
        PrintSubsequencesWhoseSumisK(arr,ind+1,n,ds,ans,s,sum);
    }
}
//*********************************Printing Subsequences whose sum is K(Only One
Subsequence)****************************/
class Solution{
    public static void main(String[] args) {
    int arr[]={1,2,1};
    int sum=2;
    ArrayList<ArrayList<Integer>> ds=new ArrayList<>();
    ArrayList<Integer> ans=new ArrayList<>();
    System.out.println(PrintSubsequencesWhoseSumisK(arr,0,3,ds,ans,0,sum));
    System.out.println(ds);
    }
    public static boolean PrintSubsequencesWhoseSumisK(int arr[],int ind,int
n,ArrayList<ArrayList<Integer>> ds,ArrayList<Integer> ans,int s,int sum) {
        //System.out.println("ps"+" "+ind+" "+n+" "+ ans+" "+ds);
        if(ind==n){
            if(s==sum){
```

```
            ds.add(new ArrayList<>(ans));
            return true;
            }
            else{return false;}
        }
        ans.add(arr[ind]);
        s+=arr[ind];
        if(PrintSubsequencesWhoseSumisK(arr,ind+1,n,ds,ans,s,sum)==true)
        {return true;}
        ans.remove(ans.size()-1);
        s-=arr[ind];
        if(PrintSubsequencesWhoseSumisK(arr,ind+1,n,ds,ans,s,sum)==true)
        {return true;}
        return false;
    }
}
//********************************Printing Subsequences whose sum is K(Only
Count)*************************************/
class Solution{
public static void main(String[] args) {
    int arr[]={1,2,1};
    int sum=2;
    ArrayList<ArrayList<Integer>> ds=new ArrayList<>();
    ArrayList<Integer> ans=new ArrayList<>();
    System.out.println(PrintSubsequencesWhoseSumisK(arr,0,3,ds,ans,0,sum));
    System.out.println(ds);
    }
    public static int PrintSubsequencesWhoseSumisK(int arr[],int ind,int
n,ArrayList<ArrayList<Integer>> ds,ArrayList<Integer> ans,int s,int sum) {
        //System.out.println("ps"+" "+ind+" "+n+" "+ ans+" "+ds);
        if(ind==n){
            if(s==sum){
            ds.add(new ArrayList<>(ans));
            return 1;
            }
            else{
                return 0;
            }
        }
        ans.add(arr[ind]);
        s+=arr[ind];
        int l=PrintSubsequencesWhoseSumisK(arr,ind+1,n,ds,ans,s,sum);
        ans.remove(ans.size()-1);
        s-=arr[ind];
        int r=PrintSubsequencesWhoseSumisK(arr,ind+1,n,ds,ans,s,sum);
        return l+r;
    }
}
//********************************Combination Sum I(Any number of
Chances)*******************************************/
//********************************Pick and Not Pick
Concept******************************************************** */
class Solution {
    public void findCombinations(int ind,int arr[],int target,List<List<Integer>>
ans,List<Integer> ds){
        if(ind==arr.length){
            if(target==0){
                ans.add(new ArrayList<>(ds));
            }
            return;
        }
        if(arr[ind]<=target){
            ds.add(arr[ind]);
            findCombinations(ind,arr,target-arr[ind],ans,ds);
            ds.remove(ds.size()-1);
        }
        findCombinations(ind+1,arr,target,ans,ds);
    }
    public List<List<Integer>> combinationSum(int[] candidates,int target){
        List<List<Integer>> ans =new ArrayList<>();
        findCombinations(0,candidates,target,ans,new ArrayList<>());
        return ans;
    }
}
//********************************Combination Sum II(Any number of
Chances)*******************************************/
//********************************Pick and Not Pick
Concept******************************************************* */
class Solution {
    public List<List<Integer>> combinationSum2(int[] candidates,int target){
        List<List<Integer>> ans=new ArrayList<>();
        Arrays.sort(candidates);
        findCombinations(0,candidates,target,ans,new ArrayList<>());
        return ans;
    }
    static void findCombinations(int ind,int[] arr,int target,List<List<Integer>>
ans,List<Integer> ds){
        if(target==0){
            ans.add(new ArrayList<>(ds));
            return;
        }
        for(int i=ind;i<arr.length;i++){
            if(i>ind && arr[i]==arr[i-1])
            {continue;}
            if (arr[i]>target)
            {break;}
            ds.add(arr[i]);
            findCombinations(i+1,arr,target-arr[i],ans,ds);
            ds.remove(ds.size()-1);
        }
    }
}
//********************************Subset Sum
I*********************************************************************/
class Solution{
    ArrayList<Integer> subsetSums(ArrayList<Integer> arr, int N){
        // code here
        ArrayList<Integer> res=new ArrayList<Integer>();
        printSubsetSums(arr,res,0,N,0);
        return res;
    }
    void printSubsetSums(ArrayList<Integer> arr,ArrayList<Integer> res,int ind,int N,int sum){
        if(ind>=N){
            res.add(sum);
            return ;
        }
        printSubsetSums(arr,res,ind+1,N,sum+arr.get(ind));
        printSubsetSums(arr,res,ind+1,N,sum);
    }
}
//********************************Subset Sum
II********************************************************************/
class Solution {
    public List<List<Integer>> subsetsWithDup(int[] nums) {
        Arrays.sort(nums);
        List<List<Integer>> ansList = new ArrayList<>();
        findSubsets(0,nums,new ArrayList<>(),ansList);
        return ansList;
    }
    public void findSubsets(int ind,int[] nums,List<Integer> ds,List<List<Integer>> ansList){
        ansList.add(new ArrayList<>(ds));
        for(int i=ind;i<nums.length;i++) {
            if(i!=ind&&nums[i]==nums[i-1])
            {continue;}
            ds.add(nums[i]);
            findSubsets(i+1,nums,ds,ansList);
            ds.remove(ds.size()-1);
        }
    }
}
//****************************** N Queen
Problem*************************************************************** */
class Solution{
        public static List<List<String>> solveNQueens(int n){
            char[][] board=new char[n][n];
            for (int i=0;i<n;i++)
                for(int j=0;j<n;j++)
                    board[i][j]='.';
            List<List<String>> res=new ArrayList<List<String>>();
            dfs(0,board,res);
            return res;
        }
        static boolean validate(char[][] board,int row,int col){
            int duprow=row;
            int dupcol=col;
            while(row>=0&&col>=0){
                if(board[row][col]=='Q')return false;
                row--;
                col--;
            }
            row=duprow;
            col=dupcol;
            while(col>=0){
                if(board[row][col]=='Q')return false;
                col--;
            }
            row=duprow;
            col=dupcol;
            while(col>=0&&row<board.length){
                if(board[row][col]=='Q')return false;
                col--;
                row++;
            }
            return true;
        }
        static void dfs(int col,char[][] board,List<List<String>> res){
            if(col==board.length){
                res.add(construct(board));
                return;
            }

            for(int row=0;row<board.length;row++){
                if(validate(board,row,col)){
                    board[row][col]='Q';
                    dfs(col+1,board,res);
                    board[row][col]='.';
                }
            }
        }
        static List<String> construct(char[][] board){
            List<String> res=new LinkedList<String>();
            for(int i=0;i<board.length;i++){
                String s=new String(board[i]);
                res.add(s);
            }
            return res;
        }
    }
//******************************Sudoku
Solver************************************************************** */
class Solution {
            public void solveSudoku(char[][] board) {
                solveSudokuUtil(board);
            }
            public boolean solveSudokuUtil(char board[][]){
                for(int i=0;i<9;i++){
```

```
                for(int j=0;j<9;j++){
                    if(board[i][j]=='.'){
                        for(char c='1';c<='9';c++){
                            if(isValid(board,i,j,c)){
                                board[i][j]=c;
                                if(solveSudokuUtil(board)==true){
                                    return true;
                                }
                                else
                                board[i][j]='.';
                            }
                        }
                        return false;
                    }
                }
            }
            return true;
        }
        public boolean isValid(char board[][],int row,int col,char ch){
            for(int i=0;i<9;i++){
                if(board[row][i]==ch){
                    return false;
                }
                if(board[i][col]==ch){
                    return false;
                }
                if(board[3*(row/3)+(i/3)][3*(col/3)+i%3]==ch){
                    return false;
                }
            }
            return true;
        }
    }
 //*******************************M-Coloring Graph
Problem****************************************************** */
class Solution{
    public boolean graphColoring(boolean graph[][], int m, int n) {
        int color[]=new int[n];
        for(int i=0;i<n;i++)
        {
            color[i] = 0;
        }
        if(graphColoringUtil(graph,m,color,0,n)==false){
            return false;
        }
        return true;
    }
    boolean graphColoringUtil(boolean graph[][],int m,int color[],int ind,int n){
        if(ind==n)
        {return true;}
        for(int c=1;c<=m;c++){
            if(isSafe(ind,graph,color,c,n)){
                color[ind]=c;
                if(graphColoringUtil(graph,m,color,ind+1,n) == true)
                    return true;
                color[ind]=0;
            }
        }
        return false;
    }
    boolean isSafe(int ind,boolean graph[][],int color[],int c,int n){
        for (int i=0;i<n;i++)
            if(graph[ind][i]&&c==color[i])
            {return false;}
        return true;
    }
}
//*********************************Palindrome
Partioning***************************************************************************** */
```

```
class Solution {
    public List<List<String>> partition(String s){
        List<List<String>> res=new ArrayList<>();
        List<String> path=new ArrayList<>();
        solve(0,s,path,res);
        return res;
    }
    public void solve(int index,String s,List<String> path,List<List<String>> res){
        if(index==s.length()){
            res.add(new ArrayList<>(path));
            return;
        }
        for(int i=index;i<s.length();i++){
            if(isPal(s,index,i)){
                path.add(s.substring(index,i+1));
                solve(i+1,s,path,res);
                path.remove(path.size()-1);
            }
        }
    }
    public boolean isPal(String s,int start,int end){
        while(start<=end){
            if(s.charAt(start)!=s.charAt(end)){
                return false;
            }
            start++;
            end--;
        }
        return true;
    }
}
//*******************************Rat in a
Maze******************************************************************** */
class Solution {
    public static void solve(int i,int j,int[][] m,int vis[][],ArrayList<String> ans,String
move,int n){
        if((i==n-1)&&(j==n-1)){
            ans.add(move);
            return;
        }
        if(i+1<n&&vis[i+1][j]==0&&m[i+1][j]==1){
            vis[i][j]=1;
            solve(i+1,j,m,vis,ans,move+"D",n);
            vis[i][j]=0;
        }
        if(j-1>=0&&vis[i][j-1]==0&&m[i][j-1]==1){
            vis[i][j]=1;
            solve(i,j-1,m,vis,ans,move+"L",n);
            vis[i][j]=0;
        }
        if(j+1<n&&vis[i][j+1]==0&&m[i][j+1]==1){
            vis[i][j]=1;
            solve(i,j+1,m,vis,ans,move+"R",n);
            vis[i][j]=0;
        }
        if(i-1>=0&&vis[i-1][j]==0&&m[i-1][j]==1){
            vis[i][j]=1;
            solve(i-1,j,m,vis,ans,move+"U",n);
            vis[i][j]=0;
        }

    }
    public static ArrayList<String> findPath(int[][] m, int n) {
        // Your code here
        int vis[][]=new int[n][n];
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                vis[i][j]=0;
            }
        }
        ArrayList<String> ans=new ArrayList<>();
        if(m[0][0]==1){
            solve(0,0,m,vis,ans,"",n);
        }
        return ans;
    }
}
//************Code 2*************Rat in a
Maze*************************************************************************** */
class Solution {
        public static void solve(int i,int j,int[][] m,int vis[][],ArrayList<String> ans,String
move,int n,int dx[],int dy[]){
            if((i==n-1)&&(j==n-1)){
                ans.add(move);
                return;
            }
            String base="DLRU";
            for(int p=0;p<4;p++){
                int nexti=i+dx[p];
                int nextj=j+dy[p];
                if(nexti>=0&&nexti<n&&nextj>=0&&nextj<n&&vis[nexti][nextj]==0&&m[nexti][nextj]==1)
{
                    vis[i][j]=1;
                    solve(nexti,nextj,m,vis,ans,move+base.charAt(p),n,dx,dy);
                    vis[i][j]=0;
                }
            }

        }
        public static ArrayList<String> findPath(int[][] m, int n) {
            // Your code here
            int vis[][]=new int[n][n];
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    vis[i][j]=0;
                }
            }
            int dx[]={1,0,0,-1};
            int dy[]={0,-1,1,0};
            ArrayList<String> ans=new ArrayList<>();
            if(m[0][0]==1){
                solve(0,0,m,vis,ans,"",n,dx,dy);
            }
            return ans;
        }
        }
//*******************************Kth Permutation
Sequence***************************************************** */
class Solution {
    public String getPermutation(int n, int k) {
        int fact=1;
        List<Integer> numbers=new ArrayList<>();
        for(int i=1;i<n;i++){
            fact=fact*i;
            numbers.add(i);
        }
        numbers.add(n);
        String ans="";
        k=k-1;
        while(true){
            ans=ans+numbers.get(k/fact);
            numbers.remove(k/fact);
            if(numbers.size()==0){
                break;
            }
            k=k%fact;
            fact=fact/numbers.size();
        }
        return ans;
    }
}
}
//Revision
//
```