Lesson Plan

Array List Class

PW SKILLS

ArrayList Class

- Before going towards the ArrayList class, let us recap about Arrays. The Array is a fixed number of groups of Similar kinds of objects placed at a continuous memory location. Similarly, ArrayList is a class which holds the object of the same kind in the order of insertion without any limit to the number of objects to store, i.e. we can say that Array is of fixed size and List is of dynamic sizing.
- ArrayList class extends AbstractList class and implements List interface.
- Syntax to create ArrayList
  List<AnyClass> list = new ArrayList<AnyClass>();
- It uses a dynamic array data structure to store objects and elements.
- It allows duplicate objects and elements.
- It maintains the insertion order.
- It is non-synchronized.
- Its elements/objects can be accessed randomly.

Methods of ArrayList

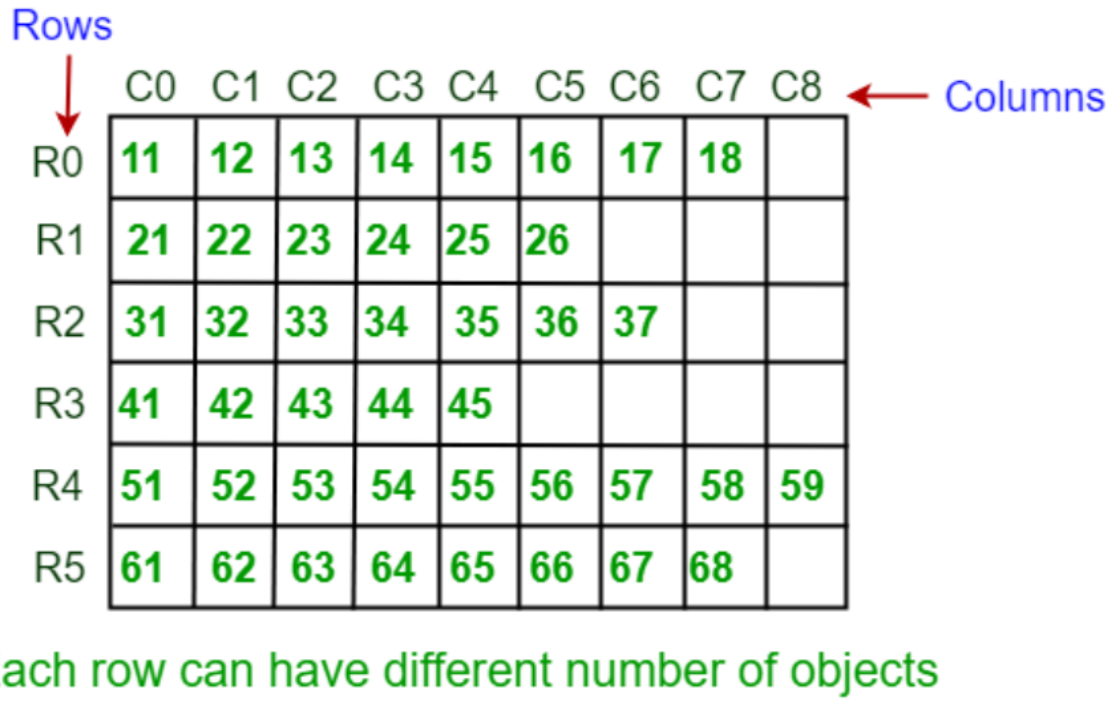
Method	Description
• void **add(**int index, E element)	• This is used to insert the specified element at the specified position in a list.
• **add**(E e)	• It is used to append the specified element at the end of a list.
• **addAll**(Collection<? extends E> c)	• It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
• **addAll**(int index, Collection<? extends E> c)	• It is used to append all the elements in the specified collection, starting at the specified position of the list.
• **clear**()	• It is used to remove all of the elements from this list.

• E remove(int index)	• It is used to remove the element present at the specified position in the list.
• boolean **remove**(Object o)	• It is used to remove the first occurrence of the specified element.
• **ensureCapacity?(int minCapacity)**	• Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument
• boolean isEmpty()	• It returns true if the list is empty, otherwise false.
• **Iterator()**	
• **listIterator()**	• Returns a list iterator over the elements in this list
• int lastIndexOf(Object o)	• It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
• Object[] toArray()	• It is used to return an array containing all of the elements in this list in the correct order.
• <T> T[] toArray(T[] a)	• It is used to return an array containing all of the elements in this list in the correct order.
• Object clone()	• It is used to return a shallow copy of an ArrayList.
• boolean contains(Object o)	• It returns true if the list contains the specified element.
• int indexOf(Object o)	• It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.

• boolean **removeAll**(Collection<?> c)	• It is used to remove all the elements from the list.
• boolean removeIf (Predicate<? super E> filter)	• It is used to remove all the elements from the list that satisfies the given predicate.
• protected void **removeRange** (intfromIndex, int toIndex)	• It is used to remove all the elements lies within the given range.
• void replaceAll (UnaryOperator<E>operator)	• It is used to replace all the elements from the list with the specified element.
• void **retainAll**(Collection<?> c)	• It is used to retain all the elements in the list that are present in the specified collection.
• E set(int index, E element)	• It is used to replace the specified element in the list, present at the specified position.
• void sort(Comparator<? super E> c)	• It is used to sort the elements of the list on the basis of the specified comparator.
• Spliterator<E> spliterator()	• It is used to create a spliterator over the elements in a list.
• List<E> subList (int fromIndex, int toIndex)	• It is used to fetch all the elements that lies within the given range.
• int size()	• It is used to return the number of elements present in the list.
• void trimToSize()	• It is used to trim the capacity of this ArrayList instance to be the list's current size.

Multidimensional Collections (or Nested Collections) is a collection of groups of objects where each group can have any number of objects dynamically. Hence, here we can store any number of elements in a group whenever we want.

Architecture of Multidimensional Collections in Java



Each row can have different number of objects

Illustration:

Single dimensional ArrayList :
[121, 432, 12, 56, 456, 3, 1023]
[Apple, Orange, Pear, Mango]

Syntax:

ArrayList <Object> x = new ArrayList <Object>();

Need for Multidimensional Collections

Unlike Arrays, we are not bound with the size of any row in Multidimensional collections. Therefore, if we want to use a Multidimensional architecture where we can create any number of objects dynamically in a row, then we should go for Multidimensional collections in java.

Syntax: Multidimensional Collections

ArrayList<ArrayList<Object>> a = new ArrayList<ArrayList<Object>>();

Illustration:

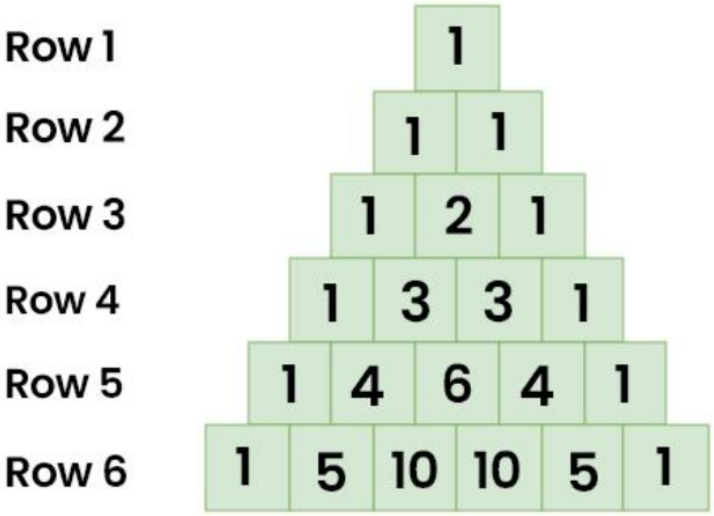
Multidimensional ArrayList: [[3, 4], [12, 13, 14, 15], [22, 23, 24], [33]]

Let us quickly peek onto add() method for multidimensional ArrayList which are as follows:

- boolean add(ArrayList<Object> e): It is used to insert elements in the specified collection.
- void add(int index, ArrayList<Object> e): It is used to insert the elements at the specified position in a Collection.

Q1. Given an integer 'numRows', generate Pascal's triangle.

The below image shows the Pascal's Triangle for N=6



Pascal's Triangle using

The number of entries in every line is equal to line number. For example, the first line has "1", the second line has "1 1", the third line has "1 2 1",.. and so on. Every entry in a line is value of a Binomial Coefficient. The value of ith entry in line number line is C(line, i). The value can be calculated using following formula.

- C(line, i) = line! / ((line-i)! * i!)

Algorithm:

- Run a loop for each row of pascal's triangle i.e. 1 to N.
  - For each row, run an internal loop for each element of that row.
    - Calculate the binomial coefficient for the element using the formula mentioned in the approach.

Below is the implementation of the above approach:

```
// Java code for Pascal's Triangle
import java.io.*;
class Main{

    // Function to print first
    // n lines of Pascal's Triangle
    static void printPascal(int n)
    {

    // Iterate through every line
    // and print entries in it
    for (int line = 0; line < n; line++)
    {
        // Every line has number of
        // integers equal to line number
        for (int i = 0; i ⩽ line; i++)
        System.out.print(binomialCoeff
                        (line, i)+" ");

        System.out.println();
    }
    }
    static int binomialCoeff(int n, int k)
    {
        int res = 1;

        if (k > n - k)
        k = n - k;

        for (int i = 0; i < k; ++i)
        {
            res *= (n - i);
            res /= (i + 1);
        }
        return res;
    }

    // Driver code
    public static void main(String args[])
    {
    int n = 7;
    printPascal(n);
    }
}
```

Output:

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1

Time complexity: O(N^3), where N is the number of rows you want to printAuxiliary Space: O(1)

Score after maximum flip:

```
class Solution {
    public int matrixScore(int[][] grid) {
        int row= grid.length;
        int col= grid[0].length;

        //  There must be 1 at the starting of every row
        for(int r=0; r<row; r++){

            if(grid[r][0] == 0){
                grid= swapRow( r, grid );
            }
        }

        //  There must be max 1's in the col.
        for(int c=0; c< col; c++){
            if(countCol1(c, grid) <= (grid.length)/2){
                grid= swapCol( c, grid );
            }

        }

        int sum= 0;
        for(int r=0; r<row; r++){

            String s= "";
            for(int i: grid[r]){
                s+= Integer.toString(i);
            }
            sum+= Integer.parseInt(s, 2);
        }
        return sum;
    }
```

```
    //  Count no. of 1's in every column
    public int countCol1(int col, int[][] grid){
        int ones= 0;
        for(int i= 0; i<grid.length; i++){

            if(grid[i][col] == 1){
                ones++;
            }
        }

        return ones;
    }

    //  Swap 1's with 0's in row
    public int[][] swapRow(int row, int[][] grid){

        for(int index= 0; index<grid[row].length; index++){

            if(grid[row][index] == 0){
                grid[row][index]= 1;
            }

            else if(grid[row][index] == 1){
                grid[row][index]= 0;
            }

        }

        return grid;
    }

    //  Swap 1's with 0's in column
    public int[][] swapCol(int col, int[][] grid){

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

            if(grid[i][col] == 0){
                grid[i][col]= 1;
            }

            else if(grid[i][col] == 1){
                grid[i][col]= 0;
            }

        }

        return grid;
    }
}
```

Q2: Leetcode 240

Solution:

```
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target)
{
        if(matrix == null || matrix.length < 1 ||
matrix[0].length <1) {
            return false;
        }
        int col = matrix[0].length-1;
        int row = 0;
        while(col ⩾ 0 && row ⩽ matrix.length-1) {
            if(target == matrix[row][col]) {
                return true;
            } else if(target < matrix[row][col]) {
                col--;
            } else if(target > matrix[row][col]) {
                row++;
            }
        }
        return false;
    }
}
```