

# Recitation 5



# Overview

---

- More 1D arrays
- The reserved word 'final'
- 2D Arrays

# Question 1 – Filter all the strings that start with given letter

---

- Given an array of strings, and a letter, return all the string which start with the given letter.
- **Note:** you may assume that the `arr.length >= 1`.
- Examples:
  - `filterByStartChar({"hello", "test", "hi"}, 'h')`
    - `{"hello", "hi"}`
  - `filterByStartChar({"hi", "cat", "apple", "no"}, 'c')`
    - `{"cat"}`
  - `filterByStartChar({"hi", "cat", "apple", "no"}, 'N')`
    - `{}`
  - `filterByStartChar({"dog", "sun", "sky"}, 's')`
    - `{"sun", "sky"}`

# Question 1 - Solution

```
public static String [] filterByStartChar(String[] nums, char ch) {  
    String [] res = new String [countStringsStartWithCh(arr, ch)];  
    int index = 0;  
    for (int i = 0; i < arr.length; i++) {  
        if (arr[i].charAt(0) == ch) {  
            res[index] = arr[i];  
            index++;  
        }  
    }  
    return res;  
}
```

```
public static int countStringsStartWithCh(String [] arr, char ch) {  
    int count = 0;  
    for (int i = 0; i < arr.length; i++) {  
        if (arr[i].charAt(0) == ch) {  
            count++;  
        }  
    }  
    return count;  
}
```

## Question 1, Expansion 1 – Filter all the strings that start with given letter

---

- Given an array of strings, and a letter, return all the string which starts with the given letter. We add ignore case
- **Note:** you may assume that the `arr.length >= 1`.
- Examples:
  - `filterByStartChar({"hello", "test", "hi"}, 'h')`
    - `{"hello", "hi"}`
  - `filterByStartChar({"hi", "Cat", "apple", "no"}, 'c')`
    - `{"Cat"}`
  - `filterByStartChar({"hi", "cat", "apple", "no"}, 'N')`
    - `{"no"}`
  - `filterByStartChar({"dog", "sun", "sky"}, 's')`
    - `{"sun", "sky"}`

# Question 1, Expansion 2 - Solution

```
public static String [] filterByStartChar(String[] nums, char ch) {  
    String [] res = new String [countStringsStartWithCh(arr, ch)];  
  
    int index = 0;  
  
    for (int i = 0; i < arr.length; i++) {  
        if (toLowerCase(arr[i].charAt(0)) == toLowerCase(ch)) {  
            res[index] = arr[i];  
            index++;  
        }  
    }  
  
    return res;  
}
```

```
public static int countStringsStartWithCh(String [] arr, char ch) {  
    int count = 0;  
  
    for (int i = 0; i < arr.length; i++) {  
        if (toLowerCase(arr[i].charAt(0)) == toLowerCase(ch)) {  
            count++;  
        }  
    }  
  
    return count;  
}
```

```
public static char toLowerCase(char ch) {  
    return ch >= 'A' && ch <= 'Z' ? (char)(ch - 'A' + 'a') : ch;  
}
```

# Question 1 – Find all the strings that start with given substring

---

- Given an array of strings, and a string, return all the string which starts with the given subtring.
- **Note:** you may assume that the `arr.length >= 1`.
- Examples:
  - `filterByStartChar({"hi hello", "test", "hi"}, 'hi')`
    - `{"hi hello", "hi"}`
  - `filterByStartChar({"hi", "cat", "catacomb", "catastrophy"}, 'cat')`
    - `{"cat", "catacomb", "catastrophy"}`
  - `filterByStartChar({"hi", "cat", "apple", "no"}, "No")`
    - `{}`
  - `filterByStartChar({"dog", "sun", "sky"}, 'snow')`
    - `{}`

# Question 1, Expansion 2- Solution

```
public static String [] filterByStartString(String[] nums, String str) {  
    String [] res = new String [countByStartString(arr, ch)];  
  
    int index = 0;  
  
    for (int i = 0; i < arr.length; i++) {  
        if (arr[i].substring(0, str.length()).equals(str)) {  
            res[index] = arr[i];  
            index++;  
        }  
    }  
  
    return res;  
}
```

```
public static int countByStartString(String [] arr, String str) {  
    int count = 0;  
  
    for (int i = 0; i < arr.length; i++) {  
        if (arr[i].substring(0, str.length()).equals(str)) {  
            count++;  
        }  
    }  
  
    return count;  
}
```

## Question 2 – Find Numbers with Even Number of Digits

---

- Given an array `nums` of integers, return how many of them contain an even number of digits.
  - `findEqualLengthNumbers ({12, 345, 2, 6, 7896})`      `\\2 (values counted: 12, 7896)`
  - `findEqualLengthNumbers ({-12, 175, 22, 2917})`      `\\3 (values counted: -12, 22, 2917)`

## Question 2 - Solution

---

```
public static int findEqualLengthNumbers(int[] nums) {  
    int count = 0;  
    for (int i = 0; i < nums.length; i++) {  
        int abs = Math.abs(nums[i]);  
        String numString = "" + abs;  
        if (numString.length() % 2 == 0) {  
            count++;  
        }  
    }  
    return count;  
}
```

## Question 3 – Find the Highest Altitude

---

- There is a biker going on a road trip. The road trip consists of  $n + 1$  points at different altitudes. The biker starts his trip on point 0 with altitude equal 0.
- You are given an integer array "gain", with length  $n$ , where  $gain[i]$  is the net gain in altitude between points  $i$  and  $i + 1$  for all  $(0 \leq i < n)$ . Return the highest altitude of a point.
- Examples
  - `highestAltitude ({-5, 1, 5, 0, -7});` // 1.
  - `highestAltitude ({-4, -3, -2, -1, 4, 3, 2});` // 0
  - `highestAltitude ({3, 2, -5, 4, -1, -2, 1});` // 5
  - `highestAltitude ({10, -10, 10, -10, 10});` // 10
  - `highestAltitude ({-2, -1, -3, -4, -2});` // 0

## Question 3 - Solution

---

```
public static int highestAltitude (int[] gain) {  
    int currHeight = 0;  
    int maxHeight = currHeight;  
    for (int i = 0; i < gain.length; i++) {  
        currHeight += gain[i];  
        if (currHeight > maxHeight) {  
            maxHeight = currHeight;  
        }  
    }  
    return maxHeight;  
}
```

## Question 4 - digitFrequency

---

- The function "digitFrequency" takes an integer n as its parameter. The function should return an array of size 10, where each index i of the array represents the number of times the digit i appears in the given integer n. The method should be able to handle negative numbers as well, treating them the same as their positive counterparts.
- For example:
  - digitFrequency(1123); \\{0, 2, 1, 1, 0, 0, 0, 0, 0, 0}
    - as the digit 1 appears twice, 2 once, and 3 once in the number 1123.
  - digitFrequency(52793299); \\{0, 0, 2, 1, 0, 1, 0, 1, 0, 3}
    - as the digit 2 appears twice, 3, 5, 7 once and 9 three times in the number 52793299.

## Question 4 - Solution

---

```
public static int[] digitFrequency(int n) {  
    String str = "" + Math.abs(n);  
    int[] appears = new int[10];  
    for (int i = 0; i < str.length(); i++) {  
        int digit = str.charAt(i) - '0';  
        appears[digit]++;  
    }  
    return appears;  
}
```

## Question 4, Expansion 1 - most frequent digit

---

- The function "mostFrequentDigit" takes an integer n as its parameter. The function should return the digit which has the most appearances in the number. In case there is a tie between more than 1, take the minimum.
- For example:
  - `mostFrequentDigit(1123);`                      `\\ 1`
  - `mostFrequentDigit(5279329);`                      `\\ 2`

## Question 3, Expansion 1 - Solution

---

```
public static int mostFrequentDigit(int n) {  
    int [] appears = digitFrequency(n);  
    int maxKey = 0;  
    for (int i = 0; i < appears.length; i++) {  
        if (appears[i] > appears[maxKey]) {  
            maxKey = i;  
        }  
    }  
    return maxKey;  
}
```

## Question 4, Expansion 2 - most frequent digit

---

- The function "biggestNumberPossible" takes a positive integer n as its parameter. The function should return the biggest possible number these combination of digits can make.
- For example:
  - `biggestNumberPossible(1123);`    `\\ 3211`
  - `biggestNumberPossible(5279329);` `\\ 9975322`

## Question 3, Expansion 2 - Solution

---

```
public static int biggestNumberPossible(int n) {  
    int [] appears = digitFrequency(n);  
    int num = 0;  
    for (int i = appears.length - 1; i >= 0; i--) {  
        while (appears[i] > 0){  
            appears[i]--;  
            num = num * 10 + i;  
        }  
    }  
    return num;  
}
```

## Recitation 5

# Class Variables and the reserved word Final



# The reserved word 'final'

---

- There are 2 main kinds of variables:
  - Non constants
  - Constant
- Up to this point we declared only non-constant variables, however, sometimes we want to define a variable whose value is fixed and can't be changed once initialize, since, its value shouldn't be changed for any reason.
- Example:
  - The English AlphaBet will always have 26 letters.
  - The location of some data that we need to access.
- This is where the word 'final' kicks in, when we declare a variable which is supposed to be a constant we want to use this word. When we do declare a variable as final, we can only assign a value once.
- When declaring constants write them in all uppercase and underscore between every new word and we saw few of them already:
  - Math.PI
  - Integer.MAX\_VALUE, Integer.MIN\_VALUE
- Arrays inner values can be changed even if it is final. The value which can't be changed is the array itself.

## Question 3, Expansion 3 - digitFrequency

---

- Earlier we solved the question digitFrequency
  - The function "digitFrequency" takes an integer n as its parameter. The function should return an array of size 10, where each index i of the array represents the number of times the digit i appears in the given integer n. The method should be able to handle negative numbers as well, treating them the same as their positive counterparts.
  - For example:
    - `digitFrequency(1123);`                      `\{0, 2, 1, 1, 0, 0, 0, 0, 0, 0\}`
      - as the digit 1 appears twice, 2 once, and 3 once in the number 1123.
    - `digitFrequency(52793299);`                      `\{0, 0, 2, 1, 0, 1, 0, 1, 0, 3\}`
      - as the digit 2 appears twice, 5 once, 7 once and 3 trice in the number 52793299.
- Alter the code which uses a final class variable NUM\_DIGITS, which represents the number of possible digits.

## Question 3, Expansion 3 - Solution

---

```
public static final int NUM_DIGITS = 10;

public static int[] digitFrequency(int n) {

    String str = "" + Math.abs(n);

    int[] appears = new int[NUM_DIGITS];

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

        int digit = str.charAt(i) - '0';

        appears[digit]++;

    }

    return appears;

}
```

## Recitation 5

# Arrays - Continued

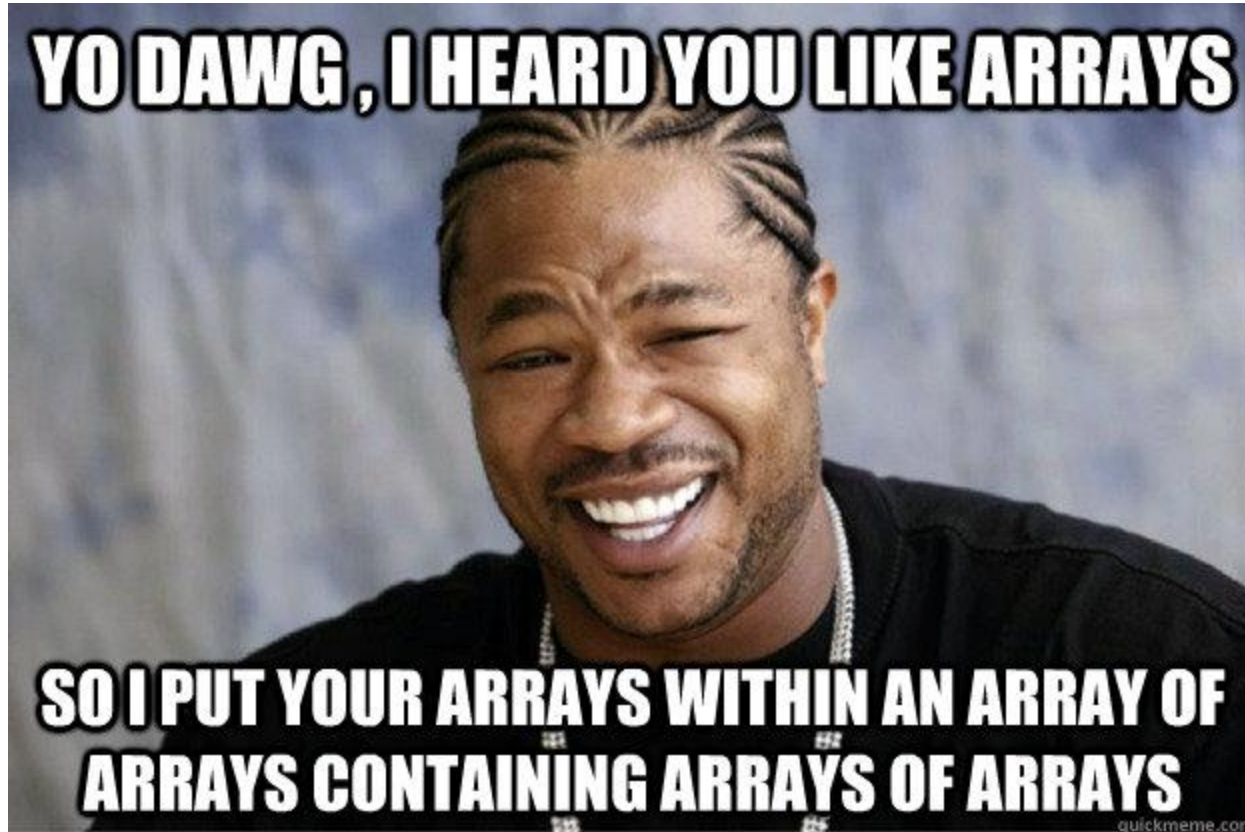
```
int array2D[3][3]={10,20,30,  
                    40,50,60,  
                    70,80,90};
```

[0]	10	20	30
[1]	40	50	60
[2]	70	80	90
	[0]	[1]	[2]

# Two Dimensional Arrays

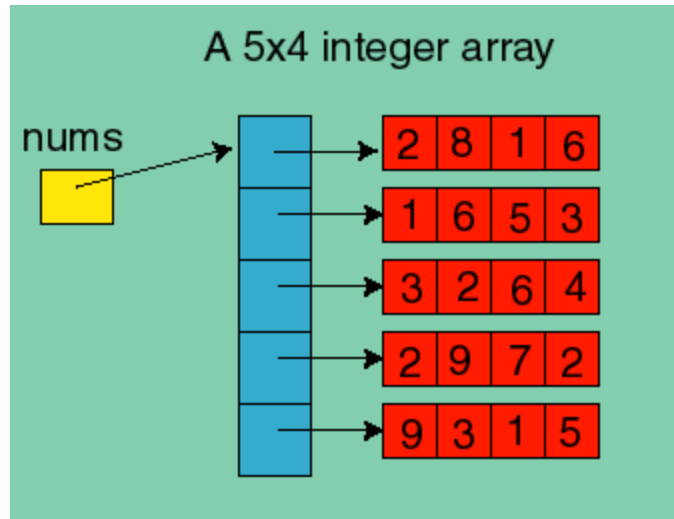
---

- Since arrays can hold any data type, why not create arrays of arrays.
- This means, every element of the array will be an array!



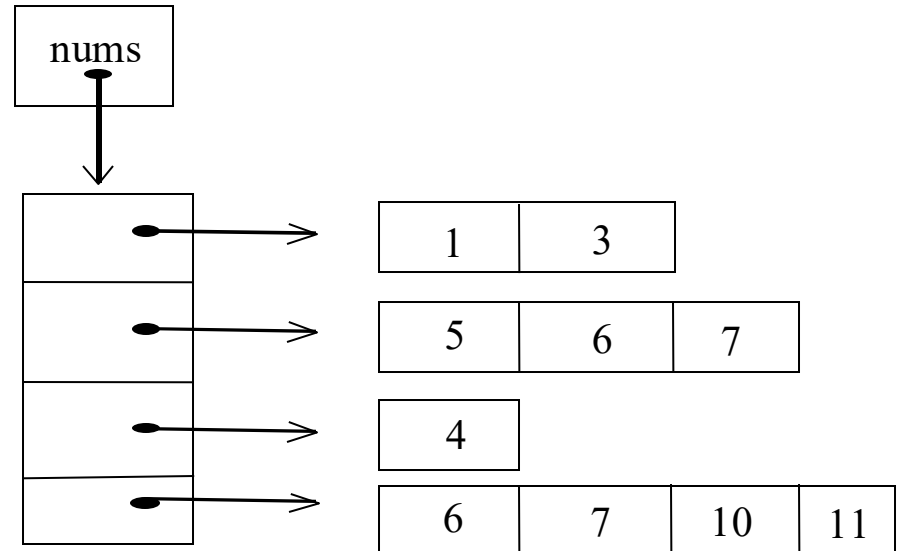
# Two Dimensional Arrays

- Every element will now have two indexes, the first corresponds to the 'outer' array, and the second to the 'inner' array.
  - `int[][] nums = new int[5][4];`



# Two Dimensional Arrays

- A two-dimensional array doesn't have to be rectangular. The inner arrays may be of varying sizes.
  - `int[][] nums = {{1, 3}, {5, 6, 7}, {4}, {6, 7, 10, 11}};`
  - `nums.length`        `//4`
  - `nums[0].length`    `//2`
  - `nums[1].length`    `//3`
  - `nums[2][2];`        `//Runtime Error`
  - `nums[1][2];`        `//7`
- **Note:** 2D arrays are not Matrices. Matrices are 2D arrays. Matrices are 2D rectangular arrays.



## Question 5 - Flattening 2D array Into 1D Array

---

- The following was a question from 2019 Midterm (Question 4)
- The function 'flatten' take rectangular 2D Array (matrix/ every row has the same length) and turn it into a 1D Array. you want to take 2D array and turn it into 1D array, contains the data of the 2D array, row after row.
- Implement the function.

## Question 5 – Solution

---

```
public static int[] flatten(int[][] arr) {  
    int[] flat = new int[arr.length * arr[0].length];  
    int index = 0;  
    for (int i = 0; i < arr.length; i++) {  
        for (int j = 0; j < arr[i].length; j++) {  
            flat[index] = arr[i][j];  
            index++;  
        }  
    }  
    return flat;  
}
```

## Question 5 - Solution, Alternative

---

```
public static int[] flatten(int[][] arr) {  
    int[] flat = new int[arr.length * arr[0].length];  
    for (int i = 0; i < arr.length; i++) {  
        for (int j = 0; j < arr[i].length; j++) {  
            flat[i * arr[0].length + j] = arr[i][j];  
        }  
    }  
    return flat;  
}
```

## Question 5, Expansion 1 - Flattening 2D array Into 1D Array

---

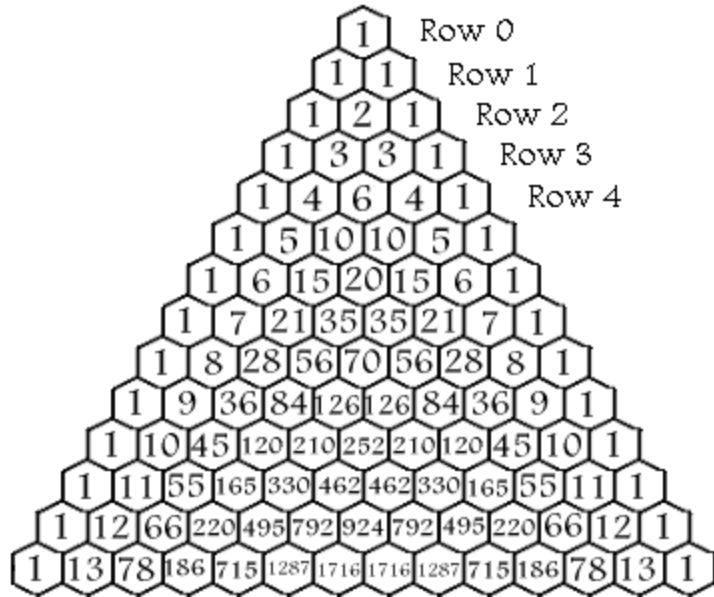
- The following was a question from 2019 Midterm (Question 6)
- Same problem but you cannot assume it is rectangular 2D Array

## Question 5, Expansion 1 - Solution

```
public static int[] flatten (int[][] arr) {  
    // the total number of elements in the 2D array  
    int numOfElems = 0;  
    for (int i = 0; i < arr.length; i++) {  
        numOfElems += arr[i].length;  
    }  
    int[] oneDArr = new int[numOfElems];  
    int index = 0; // current index in 1D array  
    for (int i = 0; i < arr.length; i++) {  
        for (int j = 0; j < arr[i].length; j++) {  
            oneDArr[index] = arr[i][j];  
            index++;  
        }  
    }  
    return oneDArr;  
}
```

## Question 6 – Pascal's Triangle

- Pascal's Triangle is of the following form:



- Starting with 1's at the edges of each row, each middle cell is the sum of the cells above it.
- Pascal's triangle has many uses in Combinatorics and number theory.

## Question 4 – Pascal's Triangle

---

- Design a program which receives a command line integer  $N > 0$  and prints the first  $N$  rows of Pascal's triangle.
  
- Before starting:
  - What will be our strategy?
  - What data are we going to store and how?
  - Any special cases we need to deal with?

## Question 6 – Pascal's Triangle (Examples)

---

```
% java Pascal 1  
1
```

```
% java Pascal 3  
1  
1 1  
1 2 1
```

```
% java Pascal 9  
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  
1 7 21 35 35 21 7 1  
1 8 28 56 70 56 28 8 1
```

## Question 6 – Solution

```
public class Pascal {  
    public static void main(String[] args) {  
        int N = Integer.parseInt(args[0]);  
        //Initialize the triangle, the triangle has N rows.  
        int[][] pascalTriangle = new int[N][];  
        //Initialize the first row separately.  
        //It has two extra spaces to avoid treating the edges differently.  
        pascalTriangle[0] = new int[3];  
        pascalTriangle[0][1] = 1;  
        //Runs through the rows of Pascal triangle.  
        for (int i = 1; i < N; i++) {  
            // row i has one more cell than i-1.  
            int lastRow = pascalTriangle[i - 1].length;  
            pascalTriangle[i] = new int[lastRow + 1];  
            for (int j = 1; j < pascalTriangle[i].length - 1; j++) {  
                pascalTriangle[i][j] = pascalTriangle[i - 1][j - 1] +  
                                       pascalTriangle[i - 1][j];  
            }  
        }  
        ...  
        // still need to print (next slide)
```

## Question 6 – Solution (Continue)

...

```
//Prints the triangle, remember each row has two extra spaces.  
//Ignore them while printing.  
for (int i = 0; i < pascalTriangle.length; i++) {  
    for (int j = 1; j < pascalTriangle[i].length - 1; j++) {  
        System.out.print(pascalTriangle[i][j] + " ");  
    }  
    System.out.println();  
}  
}  
}
```