Lecture 23

Multidimensional Arrays

Announcements for This Lecture

Material

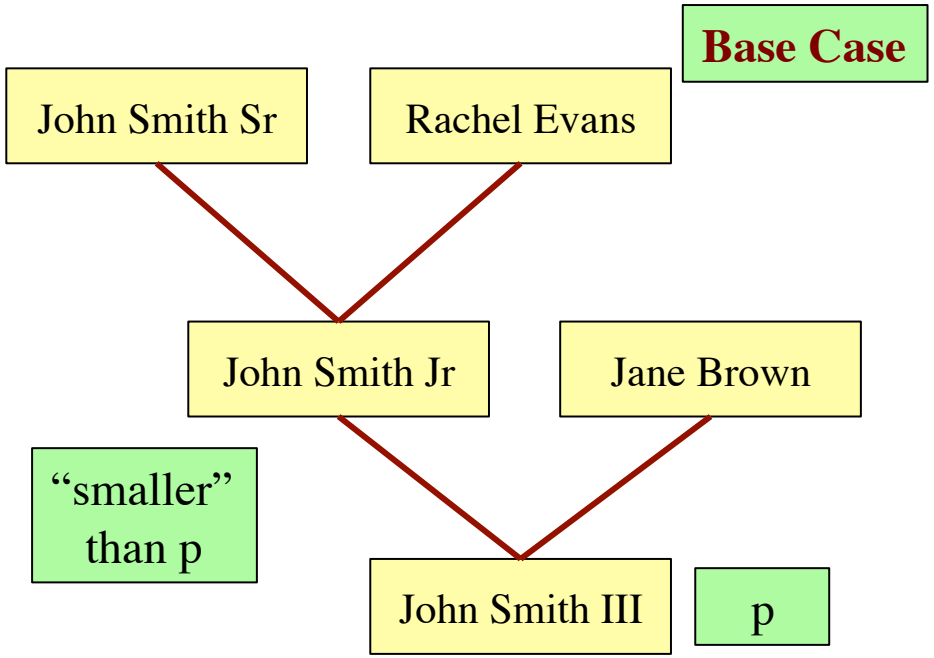
- Section 9.1
  - Last new material for final!
- Section 12.1 next time
  - Relevant to assignment
  - But not on the exam
- Next week: wrapping up
- Review sessions in 2 weeks
  - Will announce next week

Assignments

- A6 still being graded
  - Having to "eyeball it"
  - Will take us this week
- Assignment A7 now posted
  - Last assignment of semester
  - Please meet suggested dates
    - Makes it manageable
  - Due Saturday after classes

Prelim II: How I Lowered the Mean

- **Progress to Termination**
  - Arguments of recursive calls must somehow get "smaller"
  - Each call closer to base case

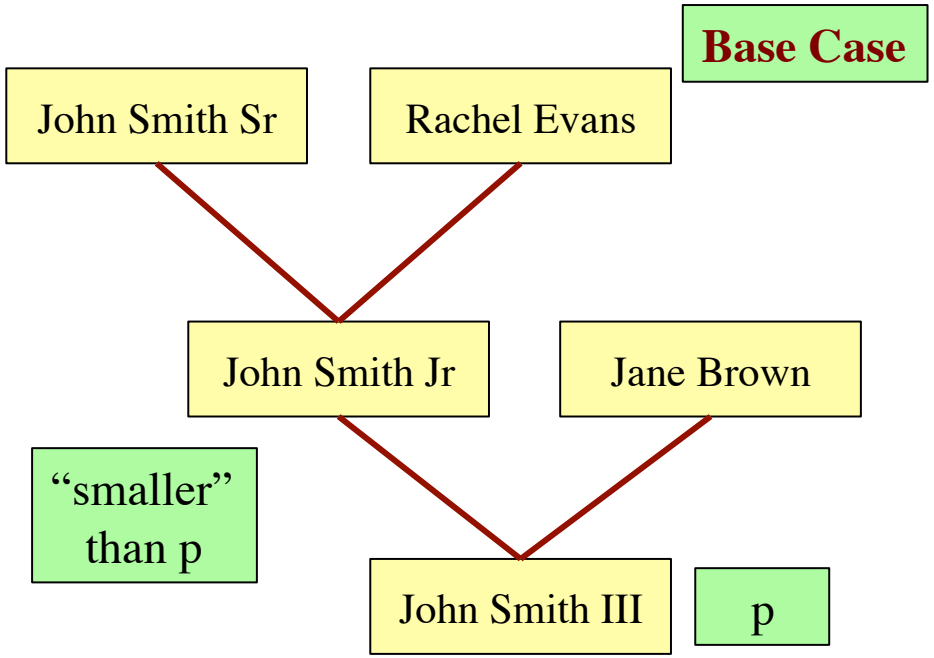



```
/** Yields: number of family members
 *  (including profile p and his/her
 *  ancestors) with given first name */
public static int withName(Profile p,
                           String s) {
  int count =
      (p.getName().equals(s) ? 1 : 0);

  if (father != null)
      count = count+withName(father,s);

  if (mother != null)
      count = count+withName(mother,s);
  return count;
}
```

Static Method

Prelim II: How I Lowered the Mean

- **Progress to Termination**
  - Arguments of recursive calls must somehow get "smaller"
  - Each call closer to base case




```
/** Yields: number of family members
 *  (including profile p and his/her
 *  ancestors) with given first name */
public int withName(String s) {
  int count =
      (getName().equals(s) ? 1 : 0);

  if (father != null)
      count = count+father.withName(s);

  if (mother != null)
      count = count+mother.withName(s);
  return count;
}
```

Instance Method

What is Up with reveal1 in A6?

Try it Yourself

```
/** Extract and return ... */
public String reveal() {
    ...
    int p= 4;
    String result= "";

    // inv: All hidden chars before
    // pixel p are in result[0..k-1]
    for (int k= 0; k < len; k= k+1) {
        result= result +
                (char) (getHidden(p));
        p= p+1;
    }

  return result;
}
```

n^2 algorithm
(n is the length of message)

```
/** Extract and return ... */
public String reveal() {
    ...
    int p= 4;
    char[] result= new char[len];

    // inv: All hidden chars before
    // pixel p are in result[0..k-1]
    for (int k= 0; k < len; k= k+1) {
        result[k]=
                (char) (getHidden(p));
        p= p+1;
    }

  return new String(result);
}
```

linear algorithm
(n time steps)

Overview of Two-Dimensional Arrays

- Type of d is **int**[][]

  ("**int** array array"/ "an array of **int** arrays")

- To declare variable d:

  int d[][];

- Create a new array and assign to d:

  d = **new int**[5][4];

- Initializer for two-dimensional array:

  int[][] d = {{5,4,7,3},{4,8,9,7},{5,1,2,3},{4,1,2,9},{6,7,8,0}};

d		0	1	2	3
	0	5	4	7	3
	1	4	8	9	7
	2	5	1	2	3
	3	4	1	2	9
	4	6	7	8	0

Overview of Two-Dimensional Arrays

- Access value in position at row 3, col 2:

  d[3][4]

- Access value in position at row 3, col 2:

  d[3][2] = 8;

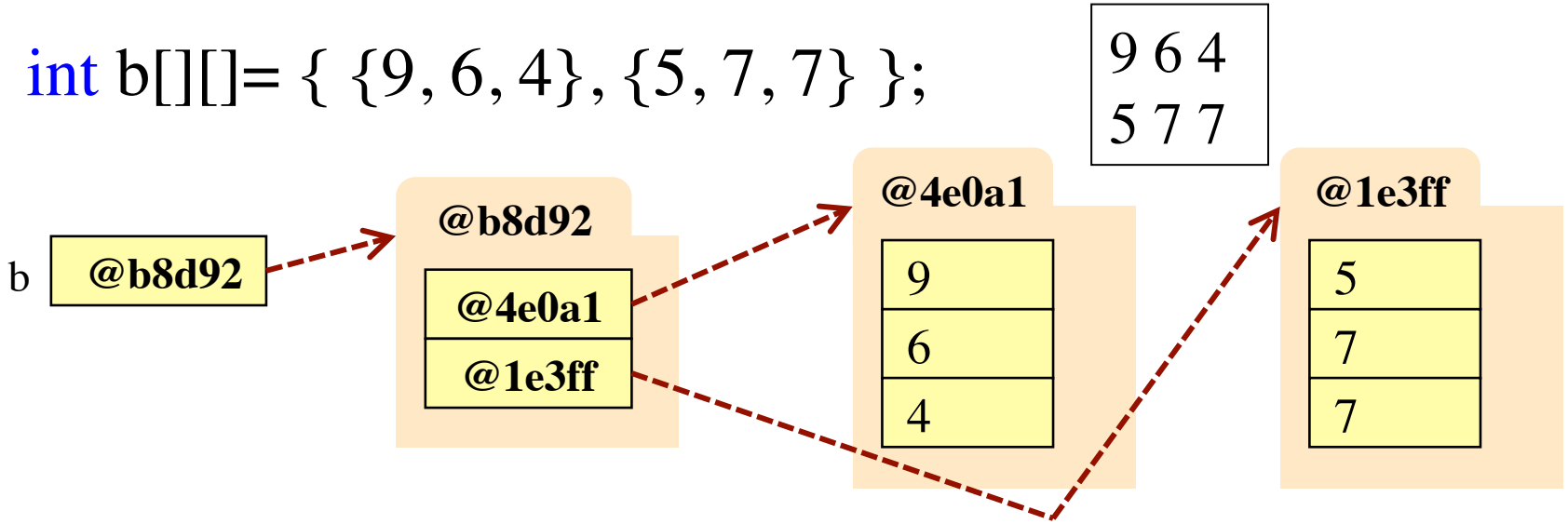
d		0	1	2	3
	0	5	4	7	3
	1	4	8	9	7
	2	5	1	2	3
	3	4	1	2	9
	4	6	7	8	0

Some Mysterious Features

- An odd symmetry
  - Number of rows of d: d.length
  - Number of columns in row r of d: d[r].length
- Also, try toString(int[]) in the demo

How Multidimensial Arrays are Stored

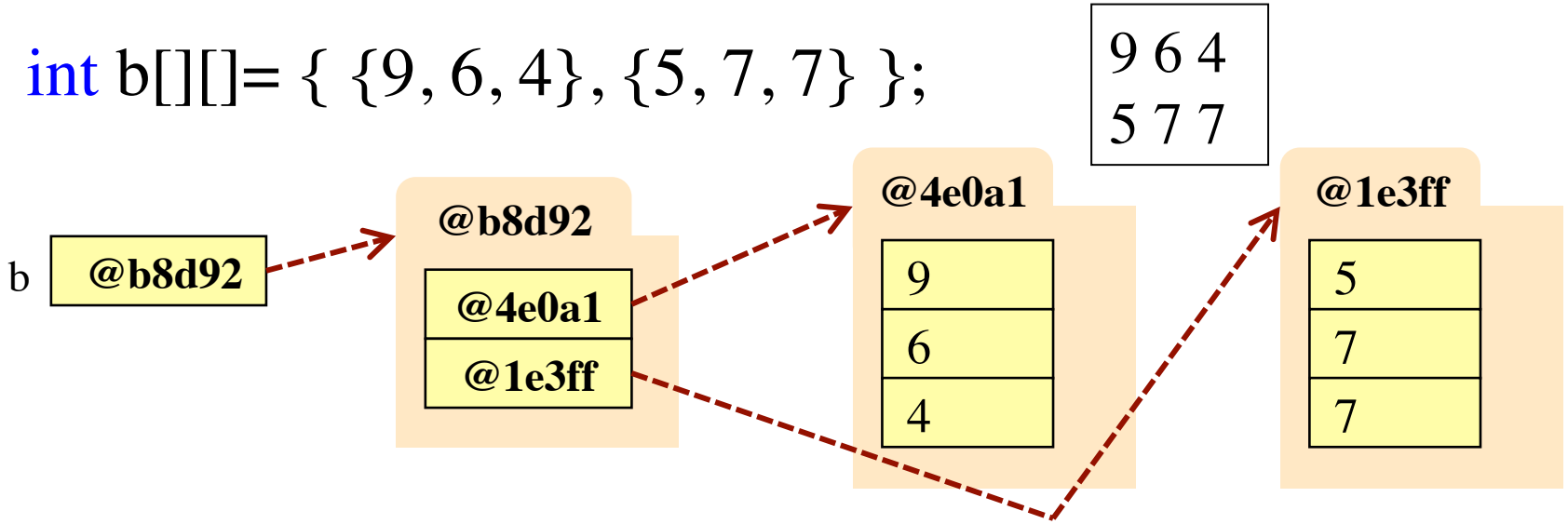
- int b[][]= { {9, 6, 4}, {5, 7, 7} };



- b holds name of a one-dimensional array object
  - Has b.length elements
  - Its elements are the names of 1D arrays
- b[i] holds the name of a one-dimensional array of **int**s
  - Has length b[i].length

How Multidimensial Arrays are Stored

- int b[][]= { {9, 6, 4}, {5, 7, 7} };



- b holds name of a one-dimensional array object
  - Has b.length elements
  - Its elements are the names of
- b[i] holds the name of a one-
  - Has length b[i].length

java.util.Arrays.deepToString recursively creates a String for all these arrays

Ragged Arrays: Rows w/ Different Length

- Declare variable b of type **int**[][]

  int[][] b;

- Create a 1-D array of length 2 and store name in b

  b= **new int**[2][] // Elements have **int**[] (and start as **null**)

- Create **int** array, store its name in b[0]

  b[0]= **new int**[] {17, 13, 19};

- Create **int** array, store its name in b[1]

  b[1]= **new int**[] {28, 95};

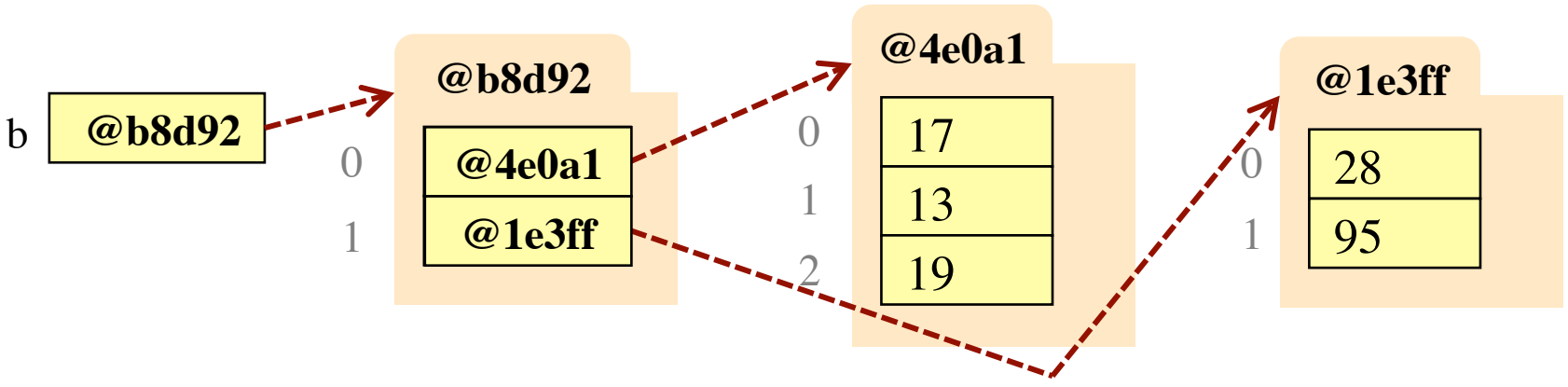
Ragged Arrays: Rows w/ Different Length

- Create **int** array, store its name in b[0]

  b[0]= **new int**[] {17, 13, 19};

- Create **int** array, store its name in b[1]

  b[1]= **new int**[] {28, 95};



Incomplete Initialization

- Sample Code:

```
int[][] b = new int[3][];
b[0] = new int[3];
b[0][0] = 2;
b[0][1] = 3;
b[1] = new int[2];
b[1][0] = 4;
b[1][1] = 5;
```

- What is b[0][2]?

A: 5
B: 0 (default int value)
C: null
D: None. Exception!
E: I don't know

Incomplete Initialization

- Sample Code:

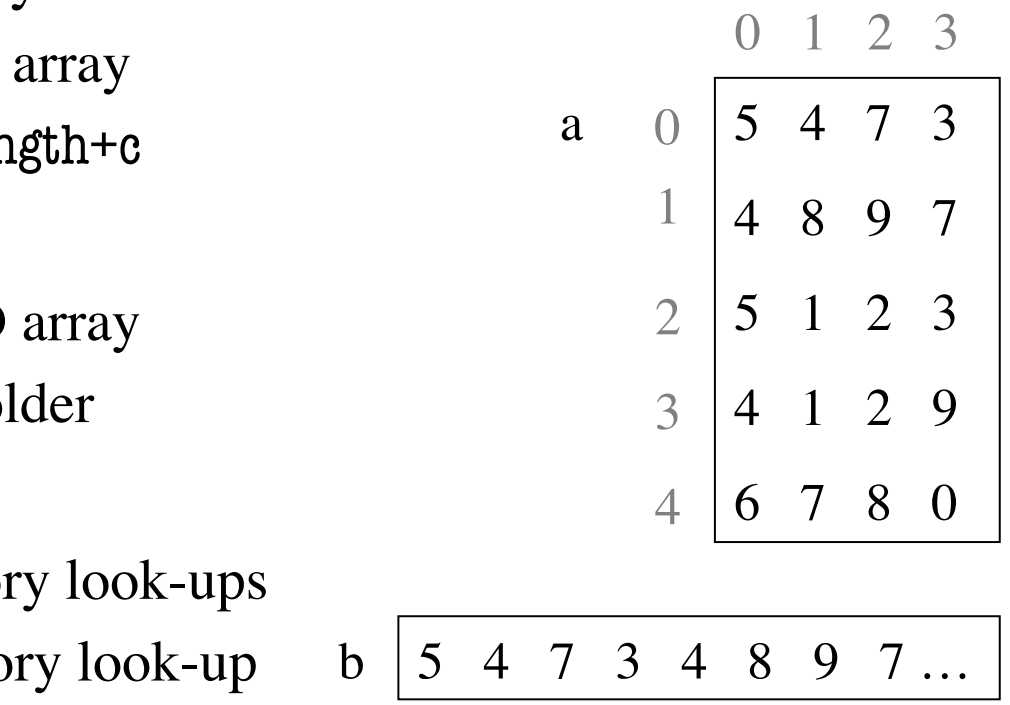
```
int[][] b = new int[3][];
b[0] = new int[2];
b[0][0] = 2;
b[0][1] = 3;
b[1] = new int[3];
b[1][0] = 4;
b[1][1] = 5;
```

- What is b[2][0]?

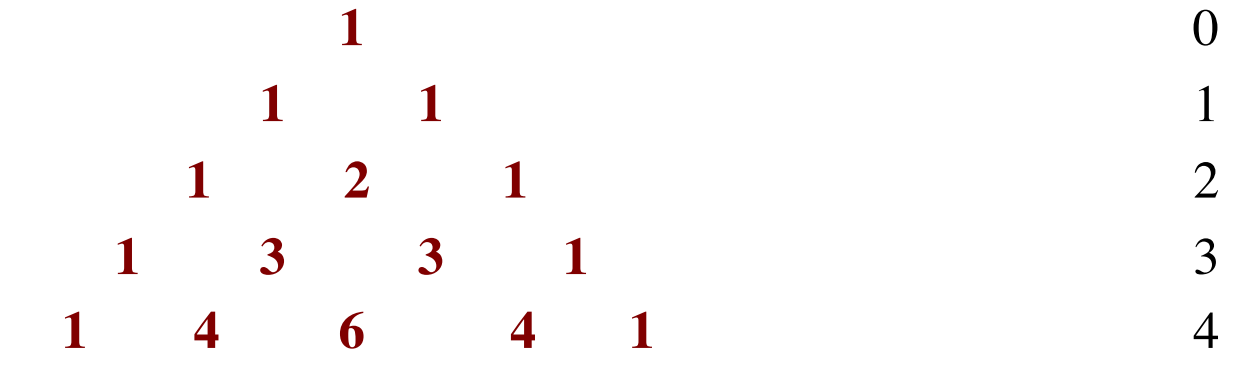
A: 5
B: 0 (default int value)
C: null
D: None. Exception!
E: I don't know

Aside: Image Array

- ImageArray used 1D array
  - Flattened version of 2D array
  - Simulated with `p = r*length+c`
- Uses less memory
  - Each row a folder in 2D array
  - ImageArray uses one folder
- Faster to access
  - 2D array needs 2 memory look-ups
  - 1D array is math+memory look-up
  - Computation faster than memory
- But 1D is harder to use

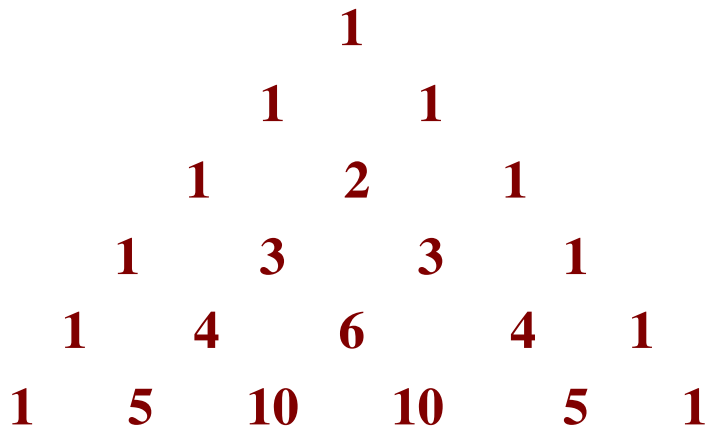


Pascal's Triangle



- Creating the triangle:
  - The first and last entries on each row are 1.
  - Each other entry is the sum of the two entries above it
  - Row r has r+1 values.

Pascal's Triangle



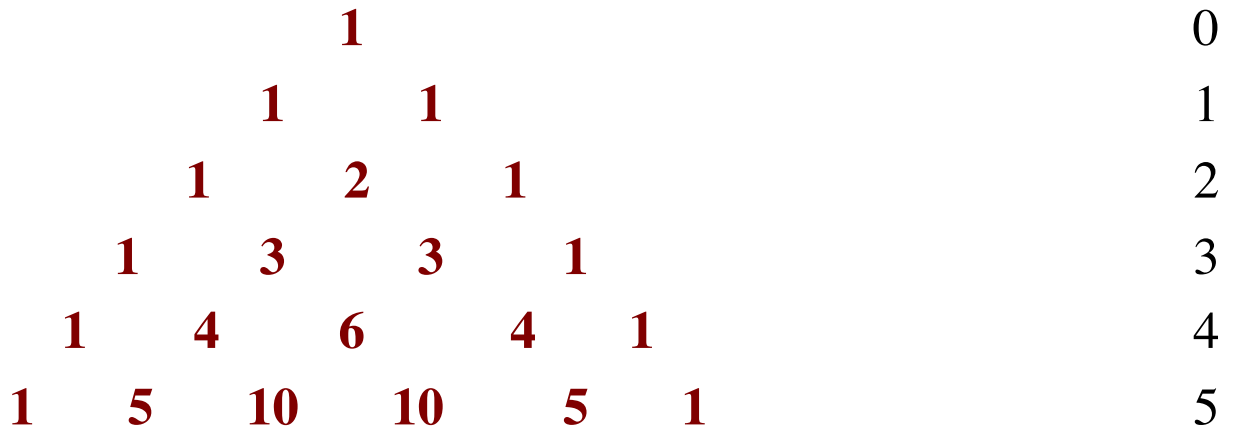
1	0
1 1	1
1 2 1	2
1 3 3 1	3
1 4 6 4 1	4
1 5 10 10 5 1	5
	…

- Entry p[i][j] = number of ways i elements can be chosen from a set of size j !
- p[i][j] = "i choose j" = $\binom{i}{j}$

Recursive formula:

for $0 < i < j$, $p[i][j] = p[i-1][j-1] + p[i-1][j]$

Pascal's Triangle



- **Binomial Theorem**: Row r gives the coefficients of $(x + y)^r$
  - $(x + y)^2 = 1x^2 + 2xy + 1y^2$
  - $(x + y)^3 = 1x^3 + 3x^2y + 3xy^2 + 1y^3$
  - $(x + y)^r = \sum_{0 \le k \le r} (k \text{ choose } r)\, x^k y^{r-k}$

Ragged Arrays for Pascal's Triangle

```
/** Yields: ragged array of first n rows of Pascal's triangle. Precondition: 0 ≤ n */
public static int[][] pascalTriangle(int n) {
    int[][] b= new int[n][];     // First n rows of Pascal's triangle
    // invariant: rows 0..i-1 have been created
    for (int i = 0; i != b.length; i=  i+1) {
        b[i]= new int[i+1];     // Create row i of Pascal's triangle
        b[i][0]=  1;            // Calculate row i of Pascal's triangle
        // invariant b[i][0..j-1] have been created
        for (int j= 1; j < i; j= j+1) {
            b[i][j]=  b[i-1][j-1] + b[i-1][j];
        }
        b[i][i]= 1;
    }
    return b;
}
```

Summing Up a Multidimensional Array

```
/** Yields: Sum of elements of b.
  *   Precondition: b is an Integer or an array with base type Integer. */
public static int sum(Object b) {
      if (b instanceof Object[]) {
         Object[] bb= (Object[]) b;
         int sum= 0;
         //inv: sum = sum of b[0..k-1]
         for (int k= 0; k < bb.length; k= k+1) {
            sum= sum + sum(bb[k]);
         }
         return sum;
      }
      // { b has type Integer }
      return 0 + (Integer) b;
}
```

Recursive call on nested array

Base Case