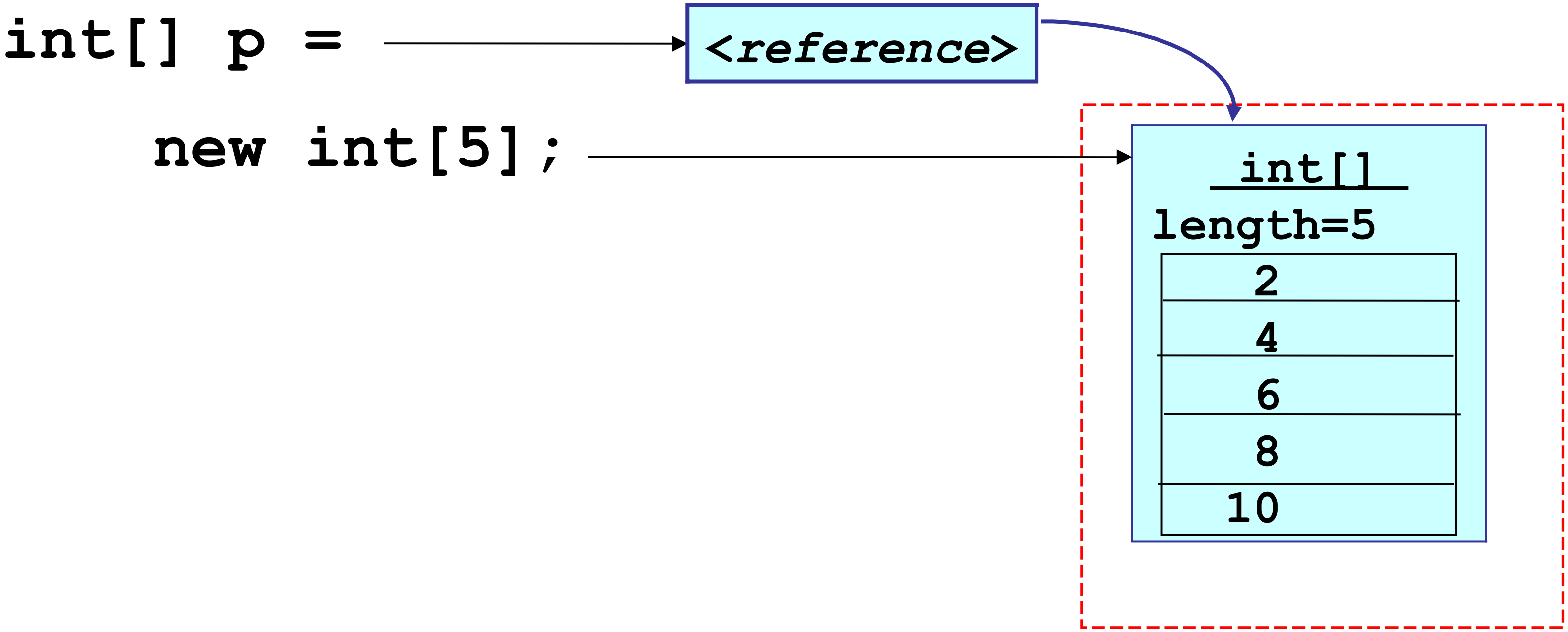
Multi-dimensional Arrays

James Brucker

1-Dimensional Arrays

- An array is a sequence of values of same type
- In Java, array is an object and knows its own length



```
int[] p =
    new int[5];
```

<reference>

int[]

length=5

2

4

6

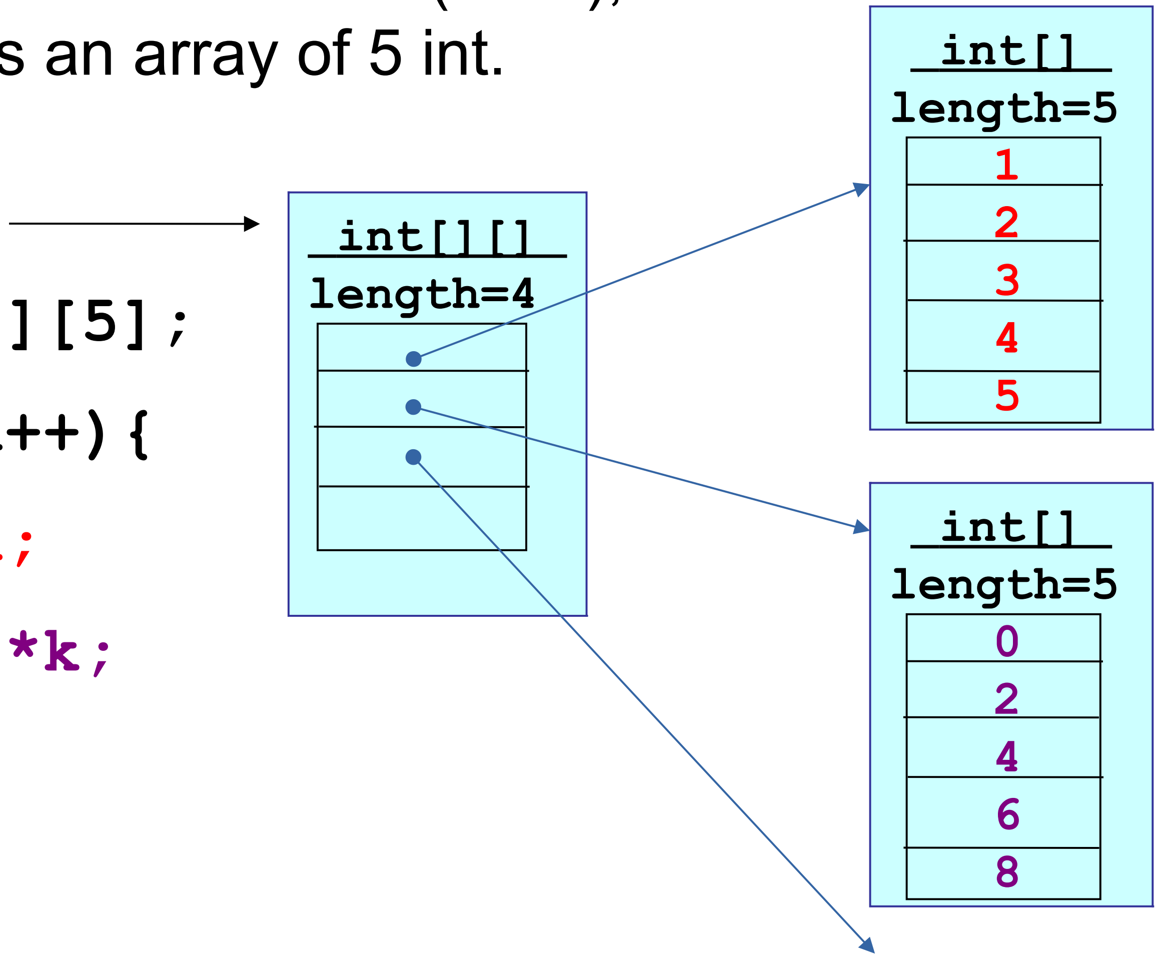
8

10

2-Dimensional Arrays

- A 2-dimensional array is an *array of arrays*
- *Example:* array of 4 elements (rows), each element is an array of 5 int.

```
int [][] m =
    new int[4][5];
for(k=0;k<5;k++){
  m[0][k] = k;
  m[1][k] = 2*k;
}
```



2-dimensional Array Syntax

1. Define a two-dimensional array reference:

```
int [][] score;
```

2. Create an array object with 4 "rows" of length 5 each:

```
score = new int[4][5];
```

1-2. Perform both steps at once:

```
int [][] score = new int[4][5];
```

3. Assign value to "row" j, element (column) k

```
score[j][k] = 999;
```

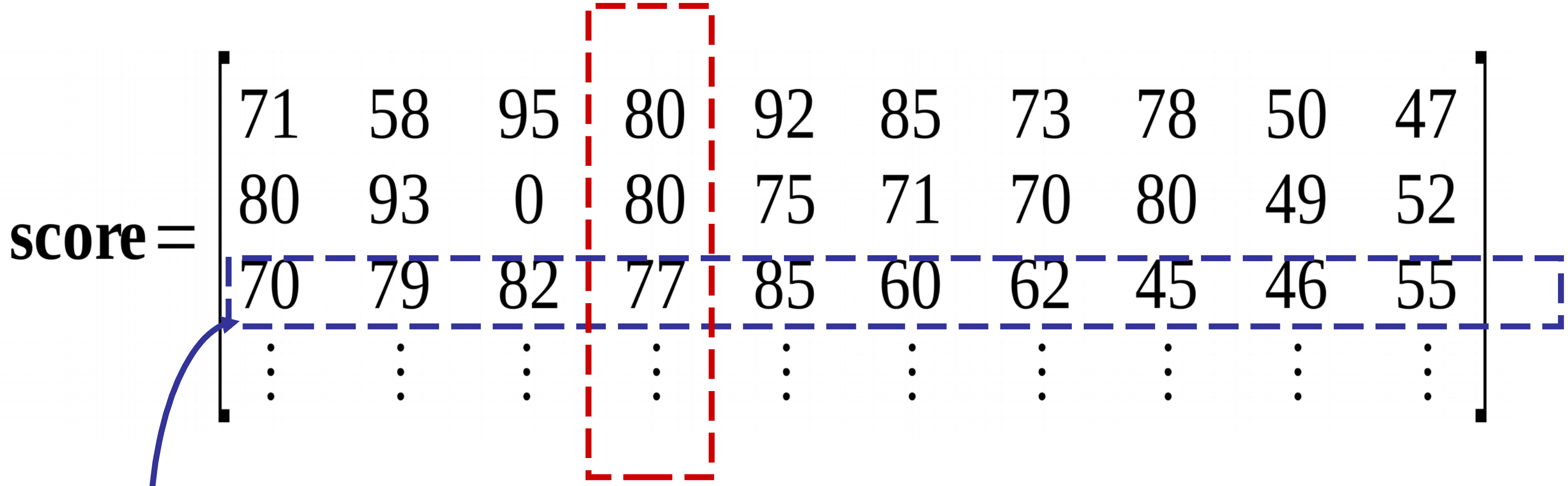
Example: student scores

`score[j]` = the scores for j-th student (an array)

```
/* score[j][k] = score of student j on lab k */
int NSTUDENT = 50;  // we have 50 students
int NLAB = 10;      // there are 10 labs
int [][] score = new int[NSTUDENT][NLAB];

/* read the lab scores */
for(int student=0; student< NSTUDENT; student++) {
    for(int lab=0; lab < NLAB; lab++)
       score[student][lab] = scanner.nextInt();
}
```

Visualize the Lab Scores



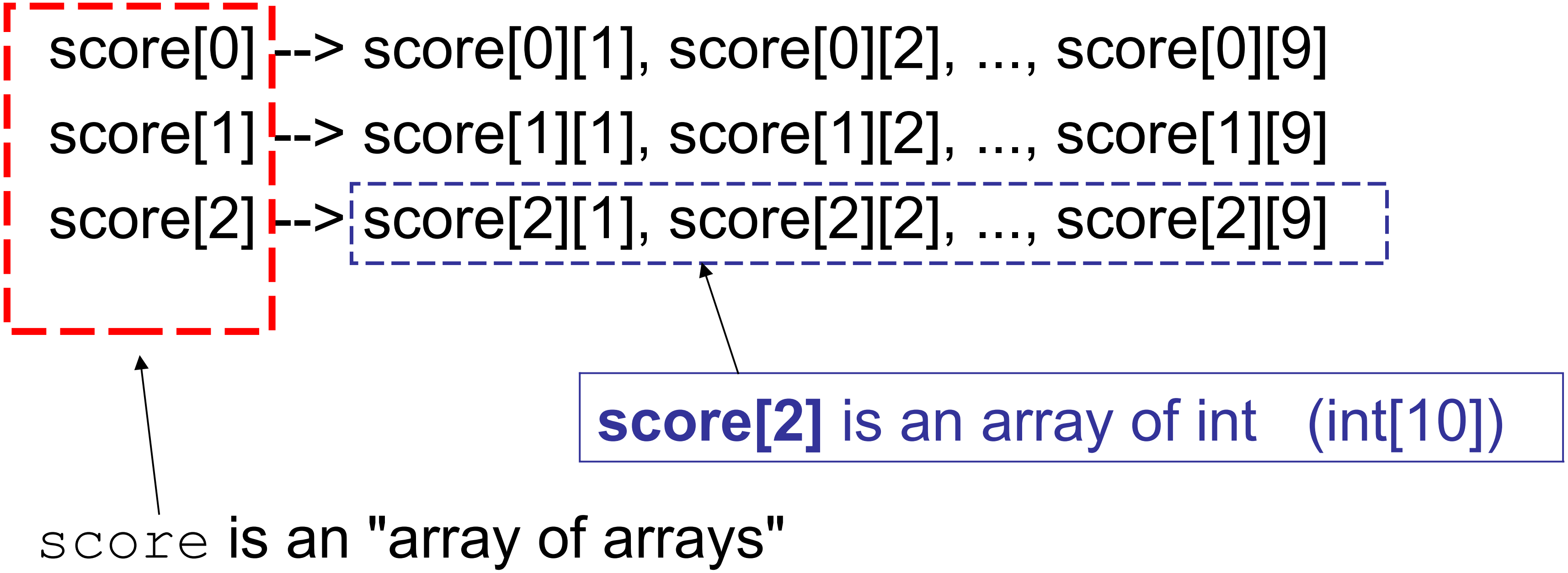
$$\mathbf{score} = \begin{bmatrix} 71 & 58 & 95 & 80 & 92 & 85 & 73 & 78 & 50 & 47 \\ 80 & 93 & 0 & 80 & 75 & 71 & 70 & 80 & 49 & 52 \\ 70 & 79 & 82 & 77 & 85 & 60 & 62 & 45 & 46 & 55 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

`score[][3]` = (column 3) scores for all students on **lab 3**.

`score[2][]` = Scores for **student 2** (an array of int)

`score[2][0]` = 70, **`score[2][1]`** = 79, ...

2-D Array in Memory



score[0] --> score[0][1], score[0][2], ..., score[0][9]

score[1] --> score[1][1], score[1][2], ..., score[1][9]

score[2] --> score[2][1], score[2][2], ..., score[2][9]

score[2] is an array of int (int[10])

`score` is an "array of arrays"

Summing Lab Scores by Student

- Sum the scores for student n:

```
int n = 8; // 9-th student (index starts at 0)
int sumScores = 0;
for(int lab=0; lab<NLAB; lab++)
     sumScores = sumScores + score[n][lab];
```

- Code Improvement: replace NLAB with the actual length of this student's scores.

```
int n = 8; // 9-th student (index starts at 0)
int sumScores = 0;
for(int lab=0; lab<__________________; lab++)
     sumScores = sumScores + score[n][lab];
```

Average scores for one lab

Find the average score on lab 5:

```
int lab = 5;
int sum = 0;
for(int j=0; j<NSTUDENT; k++)
        sum = sum + score[j][lab];
double average = ((double)sum) / NSTUDENT;
```

- Code Improvement: use actual #students in score[][]

```
int lab = 5;
int sum = 0;
for(int j=0; j<__________________; k++)
        sum = sum + score[j][lab];
double average = ((double)sum) / ______________;
```

Array length

Two-dimensional arrays have a `.length`

```
int [][] a = ...;

a.length is the number of rows in a

a[0].length is the length of row 0

a[1].length is the length of row 1
```

$$\mathbf{score} = \begin{bmatrix} 71 & 58 & 95 & 80 & 92 & 85 & 73 & 78 & 50 & 47 \\ 80 & 93 & 0 & 80 & 75 & 71 & 70 & 80 & 49 & 52 \\ 70 & 79 & 82 & 77 & 85 & 60 & 62 & 45 & 46 & 55 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

```
score.length  is 50 (rows, or students)

score[0].length is 10
```

Exercise: use `.length`

- How many students in the `score` 2-D array?

```
int[][] score = readAllScores( );
                            // all student scores
// How many students are in the array?
int numberOfStudents = score.____________ ;
```

- How many lab scores does student `n` have?

```
int n = 8;     // 9-th student
int sum = 0;
for(int lab=0; lab < _____________; lab++)
     sum = sum + score[n][lab];
```

Array as Matrix

$$a = \left[a_{row,col}\right] = \begin{array}{c} \xrightarrow{\text{columns}} \\ \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 10 & 15 & 20 \\ 10 & 20 & 30 & 40 \end{bmatrix} \end{array},\ a_{23} = 40 \quad \Big\downarrow \text{rows}$$

```
int [][] a = new int[3][4];
a[1][2] = 15;
a[0][3] = 4;
System.out.println( a.length ); // = 3
System.out.println( a[0].length ); // = 4
```

Common Array Usage

- To process every element in an array, a common usage is two nested "for" loops like this:

```
/* sum all elements in the array */
int sum = 0;
for(int row=0; row < score.length; row++) {
   for(int col=0; col < score[row].length; col++) {
      /* process element a[row][col] */
      sum = sum + score[row][col];
   }
   /* finished processing of this row */
}
```

Initializing a 2-D array

- Example: set all elements to 1

```
for(int j=0; j<a.length; j++)  /* rows */
   for(int k=0; k<a[j].length; k++)  /* cols */
        a[j][k] = 1;
```

- Example: initialize b[row][col] = row+col

```
for(int j=0; j<b.length; j++) { /* rows */
   for(int k=0; k<b[j].length; k++) { /* cols */
     // process element b[j][k]
     b[j][k] = j + k;
   }
}
```

2-D array as parameter or return

- Method with 2D array as parameter:

```
public int[] sumScore( int[][] scores ) {
```

- Return a 2D array of double:

```
public double[][] makeMatrix( int size ){
   double[][] theMatrix = new double[size][size];
   // put some values in theMatrix
   . . .
   return theMatrix;
}
```

The Hadamand Matrix

$$H=\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & \cdots \\ 1/2 & 1/3 & 1/4 & 1/5 & \cdots \\ 1/3 & 1/4 & 1/5 & & \\ 1/4 & 1/5 & & \ddots & \\ \vdots & \vdots & & & \ddots \end{bmatrix}$$

```
//TODO Write a method that returns a
//     Hadamand matrix of any size >= 1.

public _________ makeHadamand( int size)
```

The Hadamand Method

```
public double[][] makeHadamand(int size) {
   double[][] matrix = new double[size][size];

   for(int k=0; k<size; k++) {
      // be lazy -- its symmetric
      for(int j=0; j<=k; j++) {
         matrix[j][k] = matrix[k][j] = 1.0/(1+j+k);
      }
   }
   return matrix;
}
```

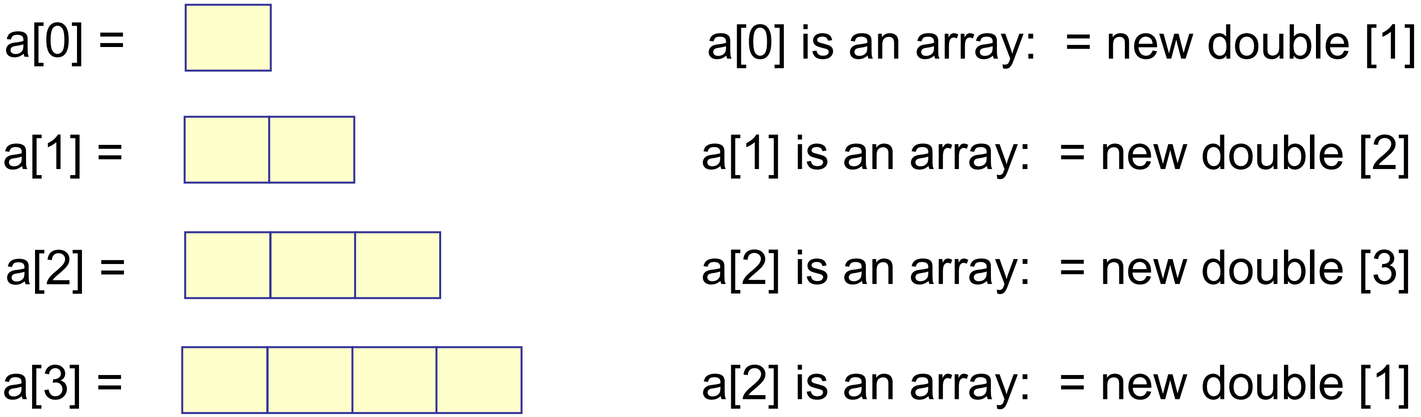
The *Truth* about 2-D Arrays

Java doesn't have 2-dimensional array!

2-D array is an array of 1-D arrays

- 2-D array in Java is really an array of arrays.
- Each row of the array is an array reference.

```
final int N = 10;
double [][] a;
a = new double[N][ ]; // create rows (an array)
for(int k=0; k<N; k++)
   a[k] = new double[k+1]; // create columns
```



a[0] = a[0] is an array: = new double [1]

a[1] = a[1] is an array: = new double [2]

a[2] = a[2] is an array: = new double [3]

a[3] = a[2] is an array: = new double [1]

Ragged Array Example

- We record the rainfall month for the days when it rains.
- How would you read this data into a 2-D array?
- How would you compute the total rainfall each month?

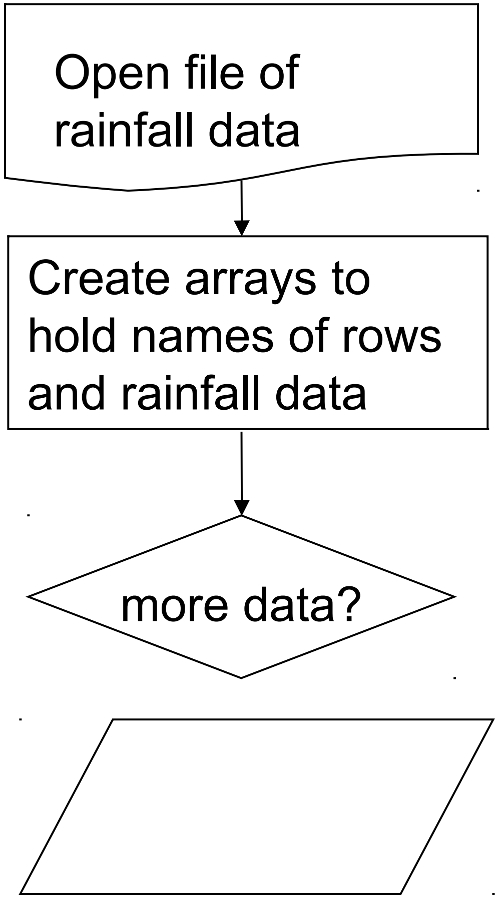
```
                    Rainfall data
jan   5 1.5 2.3 0.5 2.0 0.1
feb   4 1.1 0.3 0.3 1.0
mar   3 1.0 1.3 0.3
apr   0
may   0
jun   0
jun   0
jul   1 1.5
aug   4 0.8 1.2 1.8  0.9
sep 10  2.4 1.8 3.0 2.0 1.5 2.0 1.8 3.2 1.1 0.9
```

No rain

Output from Rainfall Problem

```
Month   Total Rain   Number of Rain days
Jan         6.4               5
Feb         2.7               4
Mar         ...              ...
```

Algorithm for Rainfall Problem



Open file of rainfall data

Create arrays to hold names of rows and rainfall data

more data?

`month []` = row names

`rain[][]` = rain each day of each month

read month[k]

read number of data points this month.

Examples of 2-D Arrays

Some extra examples. OK to skip these slides.

rowmax: find the max in each row

- rowmax(int[][] a) returns the max value from each row

$$a = \begin{bmatrix} 1 & 3 & 12 & 8 \\ 10 & 2 & 7 & 9 \\ 4 & 11 & 10 & 0 \end{bmatrix} \qquad \text{rowmax}(a) = \begin{bmatrix} 12 \\ 10 \\ 11 \end{bmatrix}$$

- in each **`row`**, find the maximum element like this:

```
/* find the largest value in this row */
max = a[row][0];
for(int col=1; col < a[row].length; col++)
    if ( a[row][col] > max ) max = a[row][col];
/* done processing this row.  save max value. */
rowmax[ row ] = max;
```

rowmax: find the max of each row (2)

- rowmax returns an array: one element for each row of a

```
public static int [] rowmax( int [][] a ) {
   int max;
   int rows = a.length;
   int [] rowmax = new int[ rows ];
   for(int row = 0; row < rows; row++) {
      /* find the largest value in this row */
      max = a[row][0];
      for(int col=1; col < a[row].length; col++)
         if ( a[row][col] > max ) max = a[row][col];
      /* record the max value for this row. */
      rowmax[ row ] = max;
   }
   return rowmax;
}
```

Pascal's Triangle

- Pascal's Triangle is a pyramid of binomial coefficients.
- Each element is the sum of 2 elements above it.

```
                    1
                 1     1
              1     2     1
           1     3     3     1
        1     4     6     4     1
     1     5    10    10    5     1
```

Pascal's triangle can be applied to combinatorial problems. It can also be used in algebra:

$$(x+y)^4 = 1x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + 1y^4$$

Pascal's Triangle (2)

Implement Pascal's Triangle as a 2-D array of size n.

1. Create a 2-D array with n rows:

```
int [][] p = new int[n];
```

2. Create each row (say, row number **`row`**)

```
p[row] = new int[row+1];
```

3. Compute elements using Pascal's rule

```
p[row][0] = p[row][row] = 1;
p[row][k] = p[row-1][k] + p[row-1][k-1];
```

Pascal's Triangle (3)

- Implement Pascal's Triangle as a 2-D array.

```
/** generate Pascal's triangle of size n rows */
int [][] Pascal( int n ) {
   // create array for row references
   int [][] p = new int[n];
   // create row = 0, 1, ..., n-1 of triangle
   for(int row=0; k < n; k++) {
      p[row] = new int[row+1];
      p[row][0] = 1;
      for(int k=1; k<p[row]; k++)
         p[row][k] = c[row-1][k] + p[row-1][k-1];
      p[row][row] = 1;
   }
   return p; // return reference to 2-D array
}
```

Vector-Matrix Multiplication

How would you multiply a 2-dimensional array **a** by a 1-dimensional array **x**?

```
/* return a vector that is the product of a*x (matrix * vector) */
public static double [] multiply( double[][] a,
   double [] x) {
   int nrows = a.length;
   int ncols = x.length;
   double [ ] y = new double[ nrows ];
   for(int i = 0; i < nrows; i++ ) {
      double sum = 0.0;
      for(int j = 0; j < ncols; j++)
         sum += a[i][j]*x[j];
      y[i] = sum;
   }
   return y;
}
```

Array Multiplication

- Let

  A = [a_{ij}] = array of size m x n

  B = [b_{ij}] = array of size n x p
- What is C = A * B ?
- What are the dimensions of C? m x p
- Formula for computing C = [c_{ij}]

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k.j}$$

Transpose an Array

- A common task to to switch the rows and columns of an array.

$$a=\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 10 & 15 & 20 \\ 10 & 20 & 30 & 40 \end{bmatrix} \xrightarrow{\text{transpose}} a^T=\begin{bmatrix} 1 & 5 & 10 \\ 2 & 10 & 20 \\ 3 & 15 & 30 \\ 4 & 20 & 40 \end{bmatrix}$$

If **`a`** is a 3 x 4 array,

then **`b = transpose(a)`** is a 4 x 3 array,

such that **`b[j][k] = a[k][j]`** for all j, k.

Transpose an Array (2)

- A **`transpose`** method must return a new array.

`int[][]`: the return value is a 2-D array of **`int`**

`int[][] a`: this parameter is a 2-D array of **`int`**

```
public static int [][] transpose( int [][] a ) {
   int rows = a.length;
   int cols = a[0].length;
   int [][] atrans = new int[cols][rols];
   .
   .
   .
   .
   return atrans;
}
```

new array for the transpose of a

return a *reference* to an **`int[][]`** array.

Transpose an Array (3)

- Inside the method we use the standard *pattern*:

```
for(int row=0; row < number_of_rows; row++)
   for(int col=0; col < number_of_cols; col++)
   process element a[row][col]
```

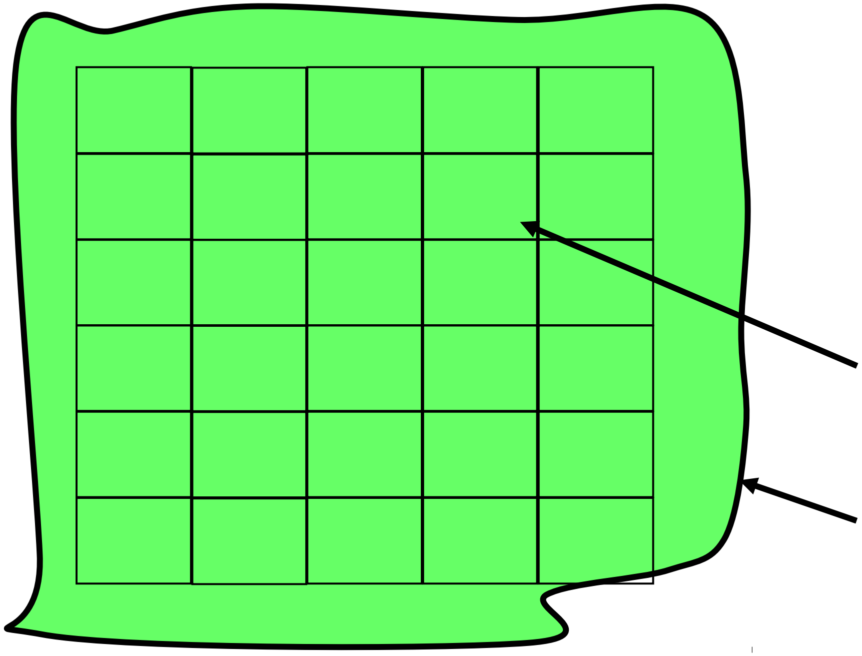
```
public static int [][] transpose( int [][] a ) {
   int rows = a.length;
   int cols = a[0].length;
   int [][] atrans = new int[cols][rols];
   for(int row = 0; row < rows; row++) {
      for(int col=0; col < cols; col++)
         atrans[col][row] = a[row][col];
   }
   return atrans;
}
```

return a *reference* to the new array.

Example: Contamination

- An environmental engineer is assessing the levels of contaminant in the soil at a polluted site. The contaminated area has been divided into a grid and the level of contaminant (C) has been measured in each rectangle in the grid.

This data can be stored as a 2D array and analysed



grid of sample locations

contaminated site

Contamination Example (2)

- A student collects the data and enters it in an array...

```
double [][] c = {
   { 0.002, 0.005, 0.004, 0.007, 0.006 },
   { 0.003, 0.001, 0.008, 0.009, 0.010 },
   { 0.002, 0.003, 0.006, 0.009, 0.008 },
   { 0.001, 0.002, 0.005, 0.008, 0.007 },
   { 0.001, 0.002, 0.004, 0.005, 0.003 },
   { 0.002, 0.001, 0.004, 0.003, 0.002 } };
```

Q: What are the dimensions of the C array?

Q: Why do we have nested parenthesis?

double [][] c = { { a, b, c}, { d, e, f}, ... { m, n, o} } ;

Contamination Example (2)'

- You can also initialize each row separately...

```
double [][] c = new double[6][]; // 6 rows
c[0] = { 0.002, 0.005, 0.004, 0.007, 0.006 };
c[1] = { 0.003, 0.001, 0.008, 0.009, 0.010 };
c[2] = { 0.002, 0.003, 0.006, 0.009, 0.008 };
c[3] = { 0.001, 0.002, 0.005, 0.008, 0.007 };
c[4] = { 0.001, 0.002, 0.004, 0.005, 0.003 };
c[5] = { 0.002, 0.001, 0.004, 0.003, 0.002 };
```

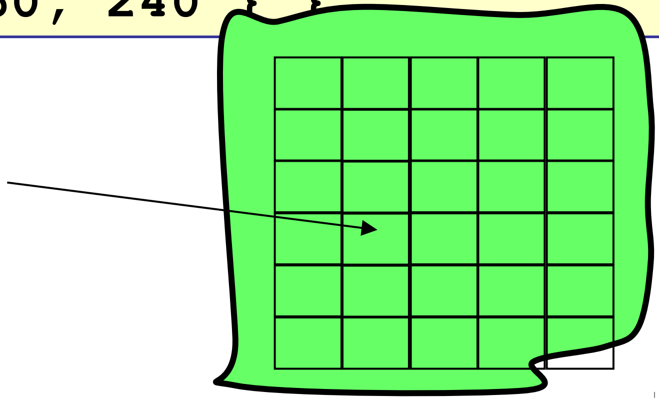
This method works even if the rows are different sizes.

Contamination Example (3)

- We have another array of data with the soil depth (in cm) in each grid cell (depth of soil down to bedrock).

```
double [][] depth = {  // dept in centimeters
   { 285, 310, 320, 315, 300 },
   { 275, 305, 310, 320, 295 },
   { 270, 300, 300, 310, 280 },
   { 260, 290, 280 ,270, 255 },
   { 255, 285, 270, 265, 250 },
   { 250, 280, 265, 260, 240 } };
```

What is the depth of this cell?



Contamination Example (4)

- The size of each cell is 2 meter by 2 meter.
  So the area of each cell is 4 m^2 = 40,000 cm^2.
- the formula for calculating from concentration (c) is:

mass = concentration * volume ;

- the volume of one cell is 40,000 * depth.
- the mass of pollutant in cell **`[j][k]`** is:

```
mass in cell [j][k] = c[j][j] * volume
= c[j][k]* ( 40000 * depth[j][k] );
```

- we need to sum this over all cells in the grid.

Contamination Example (5)

- Use nested for loops to sum the pollution over all grid cells...

```
double [][] c = { /* concentration data */ };
double [][] depth = { /* grid depth data */ };
double area = 40000; // surface area per cell
double sum = 0.0;
for (int row=0; row < c.length; row++) {
   for (int col=0; col < c[row].length; col++)
   sum += c[row][col] * area * depth[row][col];
}
// sum = total mass of pollutant
```

Building Materials

- A company makes 3 grades of cement. Each grade uses a different proportion of 4 raw materials.
- **Input**: the number of tons (1000 kg) of each product that will be produced.
- Output: how many tons of filler, binder, hardener, and sealant are needed?

	Filler	**Binder**	**Hardener**	**Sealant**
Product 1	**0.80**	**0.18**	**0.02**	**0.00**
Product 2	**0.74**	**0.20**	**0.02**	**0.04**
Product 3	**0.64**	**0.22**	**0.04**	**0.10**

Building Materials (2)

- Let amount of each product to produce be:

  prod[1] = tons of Product 1
  prod[2] = tons of Product 2
  prod[3] = tons of Product 3

- Output: tons of filler, binder, hardener, and sealant

```
filler = 0.80*prod[1] +0.74*prod[2] +0.64*prod[3]
binder = 0.18*prod[1] +0.20*prod[2] +0.22*prod[3]
harden = 0.02*prod[1] +0.02*prod[2] +0.04*prod[3]
```

	Filler	Binder	Hardener	Sealant
Product 1	0.80	0.18	0.02	0.00
Product 2	0.74	0.20	0.02	0.04
Product 3	0.64	0.22	0.04	0.10

Building Materials (3)

```
/** Compute the amount of raw materials needed to
 *  produce a given quantity of 3 products.
 *  @param product is an array of quantities of
 *         the 3 products.
 *  @return amount of raw materials needed.
 */
public double [] materials( double [] product ) {
  // mat = matrix of raw material per unit prod
  // mat[k] = { filler, binder, harden, sealant}
  //        for product k.
  double [][] mat = { {0.80, 0.18, 0.02, 0.0 },
       {0.74, 0.20, 0.02, 0.04 },
       {0.64, 0.22, 0.04, 0.10 } };
```

Building Materials (4)

```
double [][] mat = { {0.80, 0.18, 0.02, 0.0 },
                  {0.74, 0.20, 0.02, 0.04 },
                  {0.64, 0.22, 0.04, 0.10 } };
// how many raw materials are there?
int materials = mat[0].length;
// define an array for returned values
double [] quantity = new double[ materials ];
// compute the quantity of each
// raw material: sum over all products
for(int m= 0; m < materials; m++) {
    double sum = 0;
    for(int k= 0; k < product.length; k++)
       sum = sum + product[k]*mat[k][m];
    quantity[m] = sum;
}
```