Multi-dimensional Arrays

Dr. 何明昕, He Mingxin, Max

program06 @ yeah.net

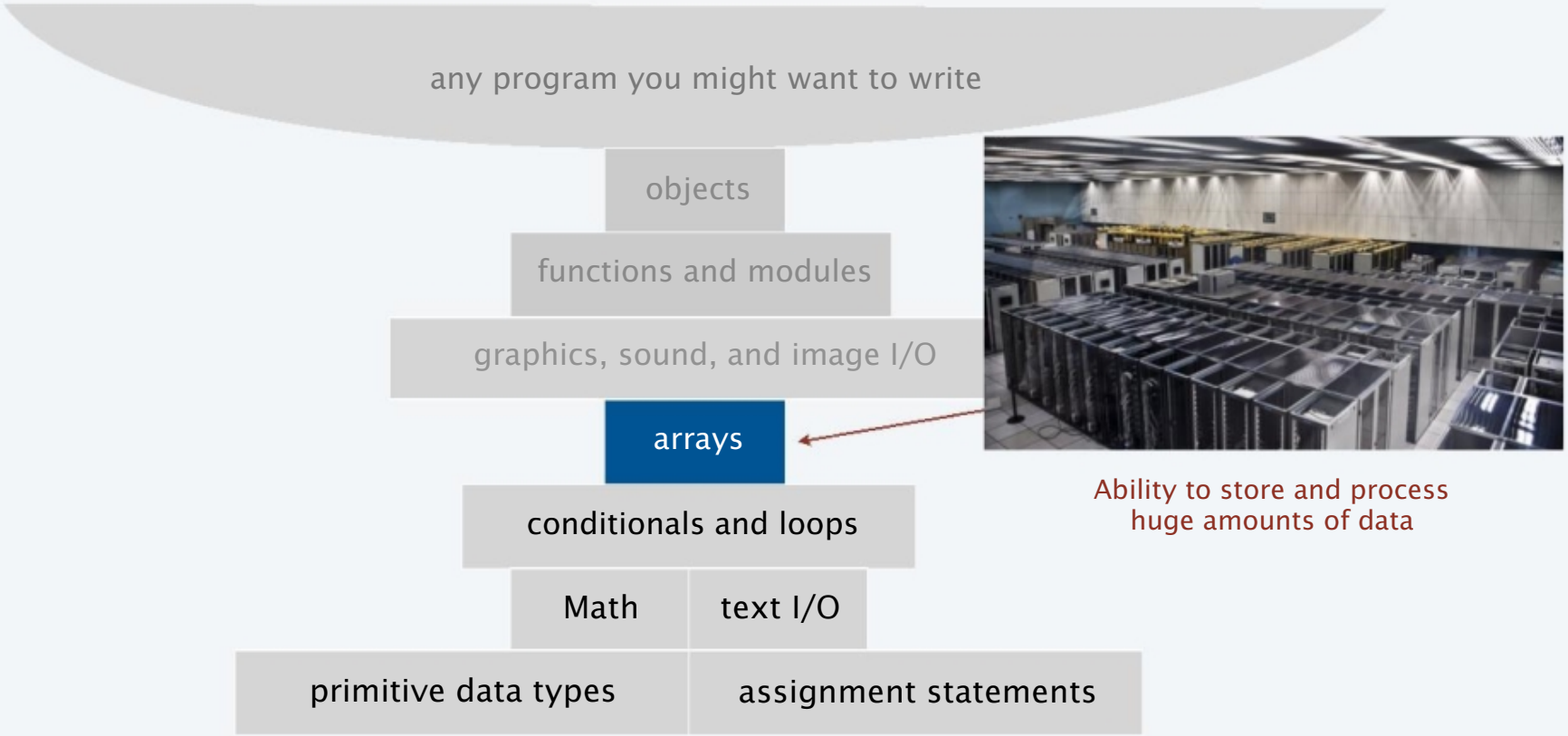
Email Subject：(AE|A2|A3) + *(Last 4 digits of ID) + Name*: *TOPIC*

Sakai: CS102A in 2018A

计算机程序设计基础

Introduction to Computer Programming

Basic building blocks for programming



Java language support for arrays

Basic support

operation	*typical code*
Declare an array	`double[] a;`
Create an array of a given length	`a = new double[1000];`
Refer to an array entry by index	`a[i] = b[j] + c[k];`
Refer to the length of an array	`a.length;`

Initialization options

operation	*typical code*
Default initialization to 0 for numeric types	`a = new double[1000];`
Declare, create and initialize in one statement	`double[] a = new double[1000];`
Initialize to literal values	`double[] x = { 0.3, 0.6, 0.1 };`

no need to use a loop like

```
for (int i = 0; i < 1000; i++)
  a[i] = 0.0;
```

BUT cost of creating an array is proportional to its length.

Arrays

- Basic concepts
- Examples of array-processing code
- Two-dimensional arrays

Two-dimensional arrays

A two-dimensional array is a *doubly-indexed* sequence of values of the same type.

Examples

- Matrices in math calculations.
- Grades for students in an online class.
- Outcomes of scientific experiments.
- Transactions for bank customers.
- Pixels in a digital image.
- Geographic data
- ...

grade

	0	1	2	3	4	5	...
0	A	A	C	B	A	C	
1	B	B	B	B	A	A	
2	C	D	D	B	C	A	
3	A	A	A	A	A	A	
4	C	C	B	C	B	B	
5	A	A	A	B	A	A	
...							

x-coordinate

Main purpose. Facilitate storage and manipulation of data.

Java language support for two-dimensional arrays (basic support)

operation	*typical code*
Declare a two-dimensional array	`double[][] a;`
Create a two-dimensional array of a given length	`a = new double[1000][1000];`
Refer to an array entry by index	`a[i][j] = b[i][j] * c[j][k];`
Refer to the number of rows	`a.length;`
Refer to the number of columns	`a[i].length;` ← can be different for each row
Refer to row *i*	`a[i]` ← no way to refer to column j

a[][] →

a[1] →

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0][5]	a[0][6]	a[0][7]	a[0][8]	a[0][9]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1][5]	a[1][6]	a[1][7]	a[1][8]	a[1][9]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]	a[2][5]	a[2][6]	a[2][7]	a[2][8]	a[2][9]

a 3-by-10 array

Java language support for two-dimensional arrays (initialization)

operation	*typical code*
Default initialization to 0 for numeric types	`a = new double[1000][1000];`
Declare, create and initialize in a single statement	`double[][] a = new double[1000][1000];`
Initialize to literal values	`double[][] p = { { .92, .02, .02, .02, .02 }, { .02, .92, .32, .32, .32 }, { .02, .02, .02, .92, .02 }, { .92, .02, .02, .02, .02 }, { .47, .02, .47, .02, .02 }, };`

no need to use nested loops like

```
for (int i = 0; i < 1000; i++)
   for (int j = 0; j < 1000; j++)
      a[i][j] = 0.0;
```

BUT cost of creating an array is proportional to its size.

Application of arrays: vector and matrix calculations

Mathematical abstraction: vector
Java implementation: 1D array

Mathematical abstraction: matrix
Java implementation: 2D array

Vector addition

```
double[] c = new double[N];
for (int i = 0; i < N; i++)
   c[i] = a[i] + b[i];
```

Matrix addition

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      c[i][j] = a[i][j] + b[i][j];
```

.30	.60	.10	+	.50	.10	.40	=	.80	.70	.50

.70	.20	.10		.80	.30	.50		1.5	.50	.60
.30	.60	.10	+	.10	.40	.10	=	.40	1.0	.20
.50	.10	.40		.10	.30	.40		.60	.40	.80

Application of arrays: vector and matrix calculations

Mathematical abstraction: vector
Java implementation: 1D array

Mathematical abstraction: matrix
Java implementation: 2D array

Vector dot product

```
double sum = 0.0;
for (int i = 0; i < N; i++)
   sum += a[i]*b[i];
```

.30	.60	.10	.	.50	.10	.40	=	.25

i	x[i]	y[i]	x[i]*y[i]	sum
0	0.3	0.5	0.15	0.15
1	0.6	0.1	0.06	0.21
2	0.1	0.4	0.04	0.25

end-of-loop trace

Matrix multiplication

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      for (int k = 0; k < N; k++)
         c[i][j] += a[i][k] * b[k][j];
```

.70	.20	.10		.80	.30	.50		.59	.32	.41
.30	.60	.10	*	.10	.40	.10	=	.31	.36	.25
.50	.10	.40		.10	.30	.40		.45	.31	.42

Pop quiz 4 on arrays

Q. How many multiplications to multiply two *N*-by-*N* matrices?

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      for (int k = 0; k < N; k++)
         c[i][j] += a[i][k] * b[k][j];
```

1. N
2. N^2
3. N^3
4. N^4

Pop quiz 4 on arrays

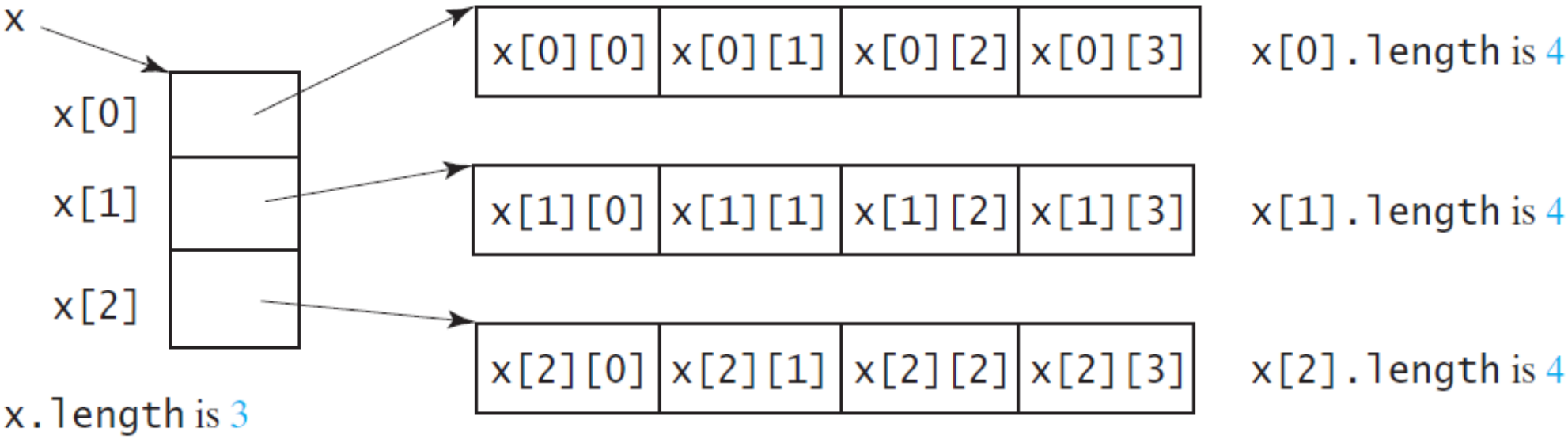
Q. How many multiplications to multiply two N-by-N matrices?

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      for (int k = 0; k < N; k++)
         c[i][j] += a[i][k] * b[k][j];
```

1. N
2. N^2
3. N^3 ← Nested forloops: $N \times N \times N$
4. N^4

Lengths of Two-dimensional Arrays

```
int[][] x = new int[3][4];
```



Lengths of Two-dimensional Arrays, cont.

```
int[][] array = {
   { 1, 2, 3 },
   { 4, 5, 6 },
   { 7, 8, 9 },
   { 10, 11, 12 }
};
```

```
array.length      4
array[0].length   3
array[1].length   3
array[2].length   3
array[3].length   3
```

```
array[4].length
```

ArrayIndexOutOfBoundsException

Ragged Arrays

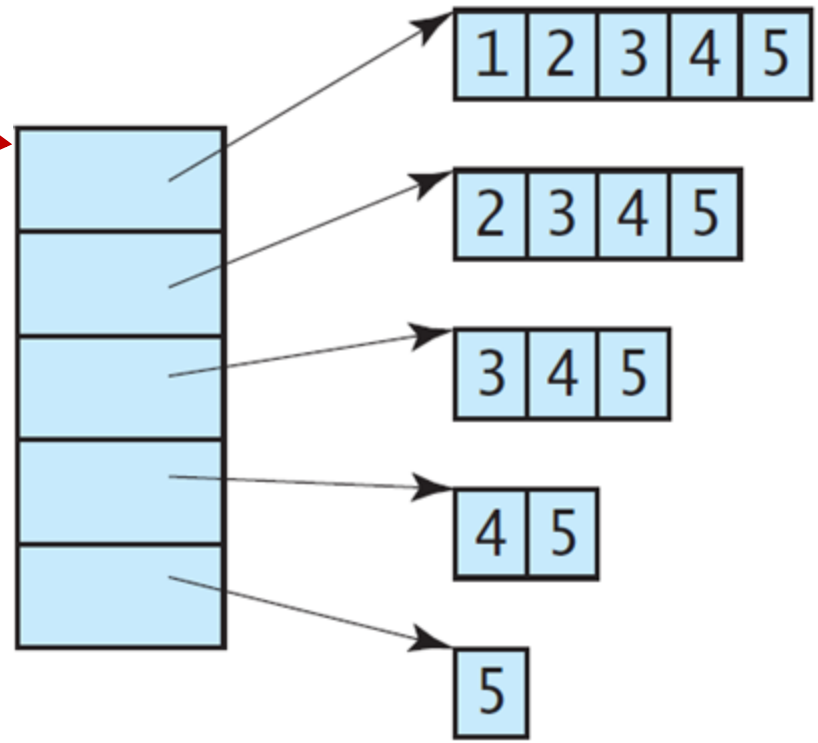
Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as *a ragged array*. For example,

```
int[][] raggedArray = {
   { 1, 2, 3, 4, 5 },
   { 2, 3, 4, 5 },
   { 3, 4, 5 },
   { 4, 5 },
   { 5 }
};
```

```
raggedArray.length is 5
raggedArray[0].length is 5
raggedArray[1].length is 4
raggedArray[2].length is 3
raggedArray[3].length is 2
raggedArray[4].length is 1
```

Ragged Arrays, cont.

```
int[][] raggedArray = {
    { 1, 2, 3, 4, 5 },
    { 2, 3, 4, 5 },
    { 3, 4, 5 },
    { 4, 5 },
    { 5 }
};
```



Processing Two-Dimensional Arrays

See the examples in the text.

1. (Initializing arrays with input values)
2. (Printing arrays)
3. (Summing all elements)
4. (Summing all elements by column)
5. (Which row has the largest sum)
6. (Finding the smallest index of the largest element)
7. (*Random shuffling*)

Initializing arrays with input values

```
java.util.Scanner input = new Scanner( System.in );
System.out.println(
  "Enter " + matrix.length + " rows and " +
  matrix[0].length + " columns: "
);
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    matrix[row][column] = input.nextInt();
  }
}
```

Initializing arrays with random values

```
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    matrix[row][column] = (int)(Math.random() * 100);
  }
}
```

Printing arrays

```
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    System.out.print( matrix[row][column] + " " );
  }

  System.out.println();
}
```

Summing all elements

```
int total = 0;
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    total += matrix[row][column];
  }
}
```

Summing elements by column

```
for (int column = 0; column < matrix[0].length; column++) {
  int total = 0;
  for (int row = 0; row < matrix.length; row++)
    total += matrix[row][column];
  System.out.println(
    "Sum for column " + column + " is " + total
  );
}
```

Random shuffling

```
for (int i = 0; i < matrix.length; i++) {
  for (int j = 0; j < matrix[i].length; j++) {
    int x = (int)(Math.random() * matrix.length);
    int y = (int)(Math.random() * matrix[i].length);
    // Swap matrix[i][j] with matrix[x][y]
    int temp = matrix[i][j];
    matrix[i][j] = matrix[x][y];
    matrix[x][y] = temp;
  }
}
```

Class Arrays and System.arraycopy

Arrays class

Provides static methods for common array manipulations.

Methods include

- sort for sorting an array (ascending order by default)
- binarySearch for searching a sorted array
- equals for comparing arrays
- fill for placing values into an array.
- toString for transfering all array elements into a String

Methods are overloaded for primitive-type arrays and for arrays of objects.

System class static arraycopy method

Copies contents of one array into another.

ArrayManipulations.java

```java
// Jhtp10, Fig. 7.22: ArrayManipulations.java
// Arrays class methods and System.arraycopy.
import java.util.Arrays;
public class ArrayManipulations {
  public static void main (String[] args) {
    // sort double array d into ascending order
    double[] d = { 8.4, 9.3, 0.2, 7.9, 3.4 };
    System.out.println( "array d : " + Arrays.toString( d ) );
    Arrays.sort( d );
    System.out.println( "sorted d : " + Arrays.toString( d ) );

    // fill 10-element array with 7s
    int[] a = new int[10];
    System.out.println( "array a : " + Arrays.toString( a ) );
    Arrays.fill( a, 7 );
    System.out.println( "filled a : " + Arrays.toString( a ) );

    // copy array c into array cCopy
    int[] c = {1, 2, 3, 4, 5, 6};
    int[] cCopy = new int[c.length];
    System.arraycopy( c, 0, cCopy, 0, c.length);
    System.out.println( "array c : " + Arrays.toString( c ) );
    System.out.println( "array cCopy : " + Arrays.toString( cCopy ) );

```

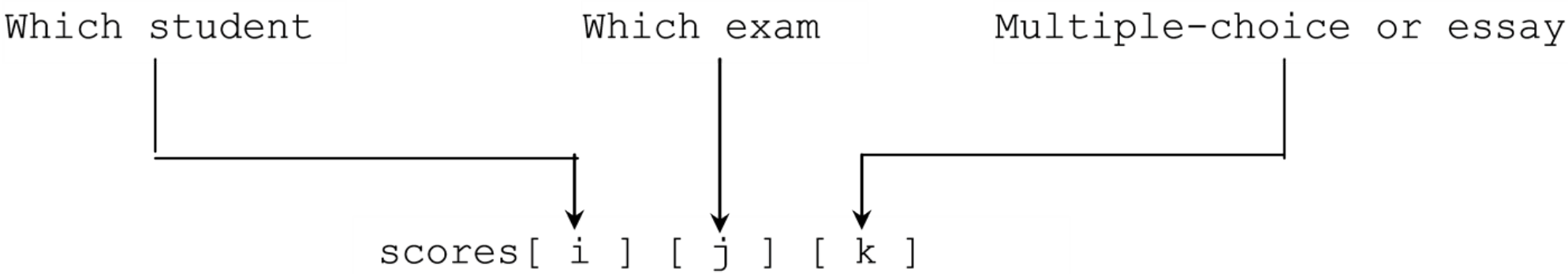
ArrayManipulations.java

```java
    // compare c and cCopy for equality
    boolean b = Arrays.equals( c, cCopy );
    System.out.printf( "c %s cCopy\n", b ? "==" : "!=" );

    // compare c and a for equality
    b = Arrays.equals( c, a );
    System.out.printf("c %s a\n", b ? "==" : "!=" );

    // search c for the value 5
    int location = Arrays.binarySearch( c, 5 );
    if (location >= 0)
      System.out.printf( "Found 5 at element %d in c\n", location );
    else
      System.out.println( "5 not found in c" );

    // search intArray for the value 8763
    location = Arrays.binarySearch( c, 8763 );
    if (location >= 0)
      System.out.printf( "Found 8763 at element %d in c\n", location );
    else
      System.out.println( "8763 not found in c" );
  }
}
```

```
H:\work\JavaProg\2018Spring\notes05>javac ArrayManipulations.java

H:\work\JavaProg\2018Spring\notes05>java ArrayManipulations
array d : [8.4, 9.3, 0.2, 7.9, 3.4]
sorted d : [0.2, 3.4, 7.9, 8.4, 9.3]
array a : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
filled a : [7, 7, 7, 7, 7, 7, 7, 7, 7, 7]
array c : [1, 2, 3, 4, 5, 6]
array cCopy : [1, 2, 3, 4, 5, 6]
c == cCopy
c != a
Found 5 at element 4 in c
8763 not found in c
```

Multidimensional Arrays

```
double[][][] scores = {
   { {7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5} },
   { {4.5, 21.5}, {9.0, 22.5}, {15, 34.5}, {12, 20.5}, {14, 9.5} },
   { {6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5} },
   { {6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5} },
   { {8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5} },
   { {9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5} }
};
```



Problem: Calculating Total Scores

Objective: write a program that calculates the total score for students in a class. Suppose the scores are stored in a three-dimensional array named `scores`. The first index in `scores` refers to a student, the second refers to an exam, and the third refers to the part of the exam. Suppose there are 7 students, 5 exams, and each exam has two parts--the multiple-choice part and the programming part. So, `scores[i][j][0]` represents the score on the multiple-choice part for the `i`'s student on the `j`'s exam. Your program displays the total score for each student.

StudentScores.java

```java
import java.util.Arrays;
public class StudentScores { // Multi-dimentional Array
  public static void main(String[] args) {
    double[][][] scores = {
      { {7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5} },
      { {4.5, 21.5}, {9.0, 22.5}, {15, 34.5}, {12, 20.5}, {14, 9.5} },
      { {6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5} },
      { {6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5} },
      { {8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5} },
      { {9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5} }
    };
    double[] total = new double[scores.length];
    for (int i = 0; i < scores.length; i++)
      for (double[] a : scores[i])  // (int j = 0; j < scores[i].length; j++)
        for (double x : a)          // (int k = 0; k < scores[i][j].length; k++)
          total[i] += x;            // += scores[i][j][k]
    System.out.print( Arrays.toString( total ) );
  }
}
```

Enhanced for statement:

```
for (type e : collection)
   statement
```

```
H:\work\JavaProg\2018Spring\notes05>javac StudentScores.java

H:\work\JavaProg\2018Spring\notes05>java StudentScores
[160.0, 163.0, 148.4, 174.4, 202.4, 181.4]
```

3. Assume that a two-dimensional array defined as following:

```
int[][] a = { {1,2,0,3}, {2,4,0,0}, {0,2,3,6}, {0,0,3,0} };
```

Write a static method named `pickPositive` to take a two-dimensional integer array `a` as parameter, and to create another new two-dimensional array that each row in the new array only contains the positive integers of the corresponding row in array `a`.

```
public static int[][] pickPositive (int[][] a) { … }
```

Please ensure that after designing the algorithm, the new array only contains 9 integers elements, which means the new array: `{{1,2,3}, {2,4}, {2,3,6}, {3}}` is what we want, and the array `{{1,2,3,0}, {2,4,0,0}, {2,3,6,0}, {3,0,0,0}}` is a wrong result. The method invoking might be something like:

```
int[][] newArray = pickPositive( a );
```

```
H:\work\JavaProg\2018Spring\notes05>javac PickPositives.java

H:\work\JavaProg\2018Spring\notes05>java PickPositives
[1, 2, 3]
[2, 4]
[2, 3, 6]
[3]
```

PickPositives.java

```java
import java.util.Arrays;
public class PickPositives { // Multi-dimentional Array
  public static void main (String[] args) {
    int[][] a = { {1,2,0,3}, {2,4,0,0}, {0,2,3,6}, {0,0,3,0} };
    int[][] newArray = pickPositive( a );
    for (int[] b : newArray)
      System.out.println( Arrays.toString( b ) );
  }

  public static int[][] pickPositive (int[][] a) {
    int[][] result = new int[a.length][];
    for (int i = 0; i < a.length; i++) {
      int[] temp = new int[a[i].length];
      int count = 0;
      for (int n : a[i])
        if (n > 0) temp[count++] = n;
      int[] row = new int[count];
      System.arraycopy( temp, 0, row, 0, count );
      result[i] = row;
    }
    return result;
  }
}
```