Array Applications

- Given a list of test scores, determine the average, maximum and minimum scores.
- Read in a list of student names and rearrange them in alphabetical order (sorting).
- Track the ups and downs of a stock index.
- Represent and analysis a digital image as a 2D array.

Array Declaration

- An array is a collection of homogenous data objects.
- Syntax
  - **DataType**[ ] *nameOfVariable*;
    - DataType: The data type of array elements
    - [ ]: We need to write a pair of square brackets next to the data type when declaring an array

Creating an Array

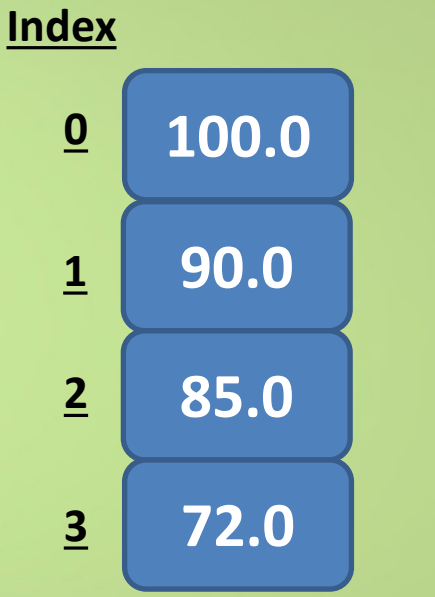
- Some examples of array declarations:
  - double[ ] testScores;
  - int[ ] studentID;
  - double[ ] stockIndex;
- The above declarations do not actually create an array. They declare a reference variable to an array.
- To create an array:
  - testScores = new double[100];
  - studentID = new int[50];
  - double[ ] stockIndex = new double[365]; // create an array
    // during declaration

Initializing an Array

- Declare, define and initialize an array using a single statement:
  - double[ ] testScore = {100.0, 90.0, 85.0, 72.0 };
- This shorthand initialization **must be in ONE statement**
  - For example, the following is wrong:

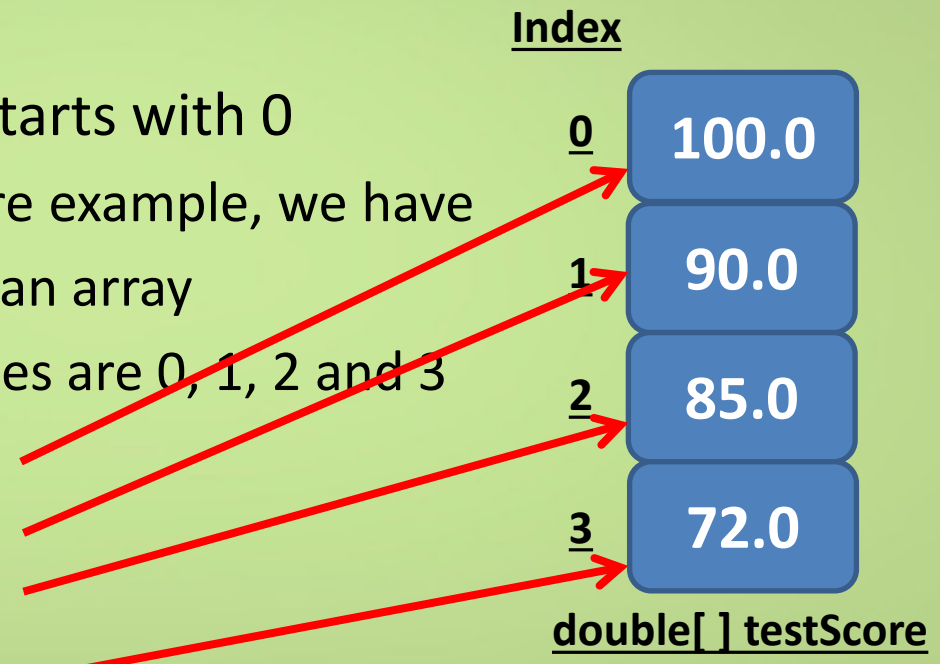
    double[ ] testScore;

    testScore = {100.0, 90.0, 85.0, 72.0};

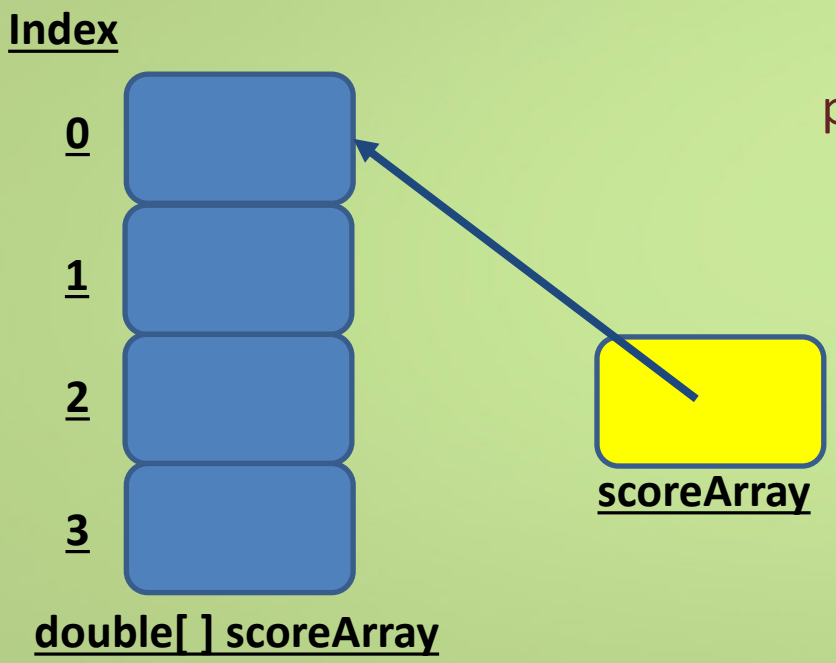


Access an array element

- The array elements are accessed through indices
- The first index starts with 0
  - In the testScore example, we have 4 elements in an array
  - The valid indices are 0, 1, 2 and 3
    - testScore[0]
    - testScore[1]
    - testScore[2]
    - testScore[3]

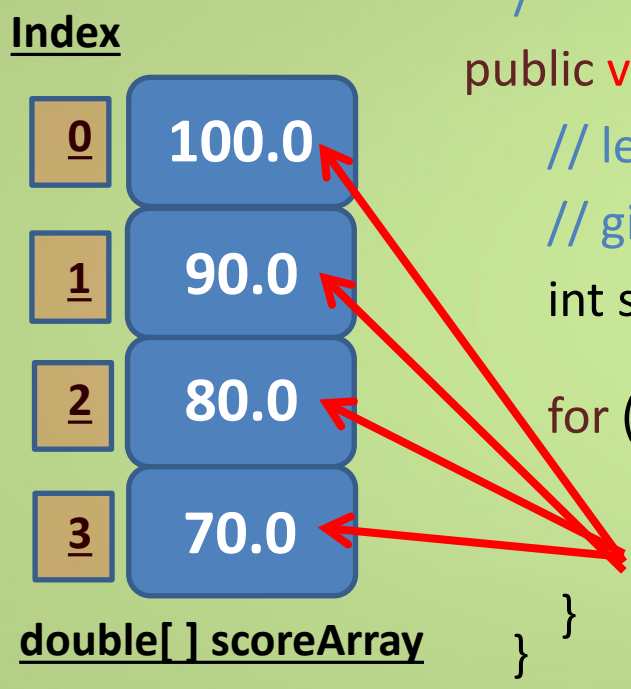


Example



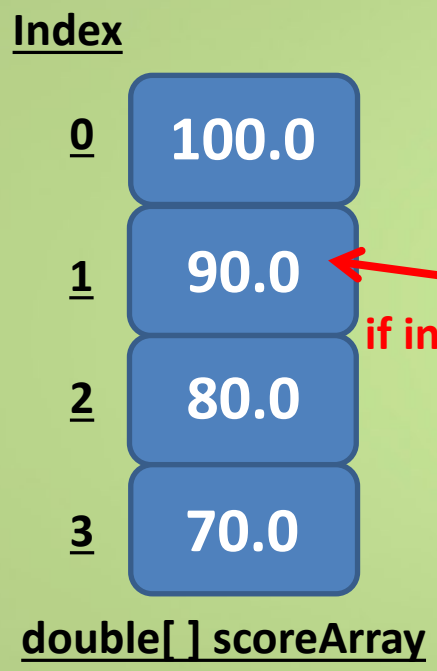

```
/**
 * Create an array for storing a set of scores
 * /
public class Scores {
  double[ ] scoreArray; // declare a reference
                        // variable to an array
  public Scores (int size) {
      scoreArray = new double[size];
  }
```

Example: setScore




```
/*
 * SetScore asks user to enter score for each array element
 */
public void setScore ( ) {
    // length is a constant instance variable for each array that
    // gives the number of elements in the array.
    int size = scoreArray.length;

    for ( int i = 0; i < scoreArray.length; i++ ) {
        IO.output("Enter score for student " + i + ": ");
        scoreArray[i] = IO.inputDouble( );
    }
}
```

Example: getScore




```
/*
 * getScore retrieves the value of an element of the array
 */
public double getScore( int index ) {
    // check to make sure that index is within 0 and array size -1
    if ( index >= 0 && index < scoreArray.length )
        return scoreArray[index];
    else {
        IO.outputln("Error: index out of range");
        return -1;
    }
}
```

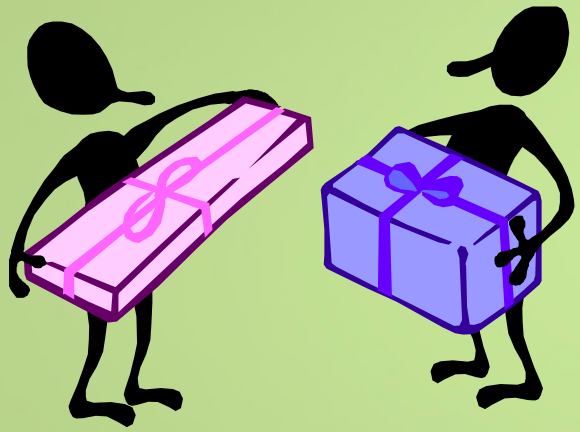
Example: Compute Average

```
/*
 * aveScore computes the average of the values in an array
 */
public double aveScore( ) {
    double sum = 0;      // for storing the cumulative sum
    int size = scoreArray.length;  // size of the array

    for (int i = 0; i < size; i++)
        sum = sum + scoreArray[i];

    return sum / size;
  }
```

Example: Find Maximum

```
/*
 * maxIndex finds the location of the largest values in an array
 * up to index size - 1
 */
public int maxIndex(int size){
    int mIndex = 0;   // index for the current maximum
    if (size > scoreArray.length) size = scoreArray.length;

    for (int i = 0; i < size; i++) {
        if (scoreArray[i] > scoreArray[mIndex]) mIndex = i;
    }
    return mIndex;
}
```

Swapping



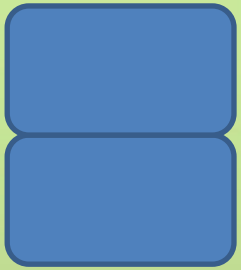
Index	double[ ] scoreArray
0	100.0
1	90.0
2	80.0
3	70.0

Swapping

Index	double[ ] A
0	100.0
1	90.0
2	80.0
3	70.0

```
public void badSwap (double n1, double n2) {
    n1 = n2;
    n2 = n1;
}
```

```
badSwap ( A[1] , A[2] );
```



Swapping

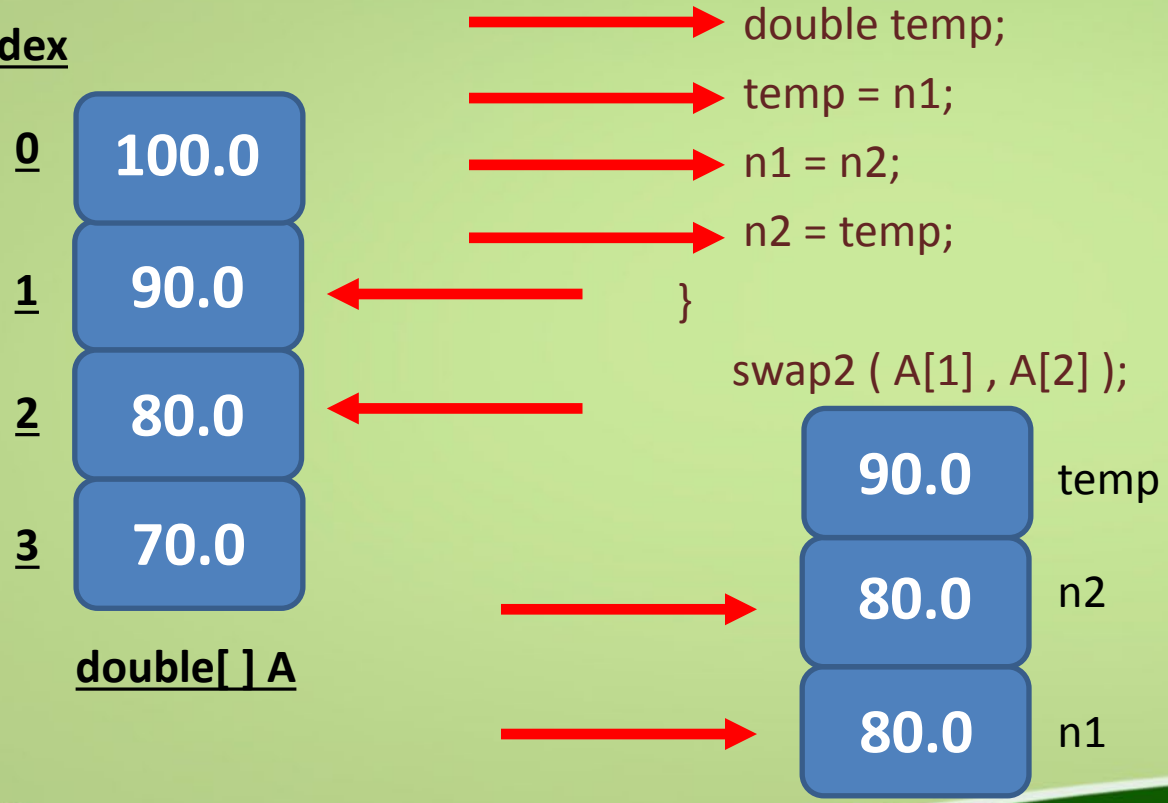

```
public void badSwap (double n1, double n2) {
    n1 = n2;
    n2 = n1;
}
```

```
badSwap ( A[1] , A[2] );
```

Swapping

```
public void swap2 (double n1, double n2) {
    double temp;
    temp = n1;
    n1 = n2;
    n2 = temp;
}
```

```
swap2 ( A[1] , A[2] );
```



Swapping


```
public void swap (double[ ] arr, int index1, int index2) {
    double temp;                    // double temp;
    temp = arr[index1];             // temp = n1;
    arr[index1] = arr[index2];      // n1 = n2;
    arr[index2] = temp;             // n2 = temp;
}
```

swap (A, 1, 2);

Scores - ArrayExample

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Close

Source Code

```
 * Demostrate the use of array for storing and manipulating a set of scores
 */
public class Scores
{
   double[] scoreArray; // declare a reference variable to an array

   public Scores(int size) {
       scoreArray = new double[size];
    }
   /**
    * SetScore asks user to enter score for each array element
    */
   public void setScore( ) {
       // length is a constant instance variable that gives the number
       // of elements in an array.
       int size = scoreArray.length;

       for (int i = 0; i < size; i++) {
           IO.output("Enter score for student " + i + ": ");
           scoreArray[i] = IO.inputDouble();
        }
```

Scores - ArrayExample

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Close

Source Code

```
public double getScore(int index) {
    if (index >=0 && index < scoreArray.length)
       return scoreArray[index];
    else {
       IO.outputln("Error: index out of range");
       return -1;
     }
}

/**
* aveScore computes the average of the values in an array
*/
public double aveScore(){
    double sum = 0;  // for storing the cumulative sum
    int size = scoreArray.length; // size of the array
    // update the cumulative sum by going all elements in the array
    for (int i = 0; i < size; i++)
        sum = sum + scoreArray[i];
    // compute and return the average score
    return sum / size;
 }
```

Scores - ArrayExample

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Close

Source Code

```
 * maxIndex finds the location of the largest values in an array up to index size - 1
 */
public int maxIndex(int size){
    int maxI = 0;  // index of the current maximum
    // check if size is larger than the maximum size of the array
    if (size > scoreArray.length) size = scoreArray.length;
    for (int i = 0; i < size; i++) {
        // check if there is a new maximum and update the index accordingly
        if (scoreArray[i] > scoreArray[maxI]) maxI = i;
    }
    // return the index of the maximum element
    return maxI;
 }
/**
 * swap two elements of the array arr indexed by index1 and index2
 */
public void swap (double[] arr, int index1, int index2) {
    double temp;
    temp = arr[index1];
    arr[index1] = arr[index2];
    arr[index2] = temp;
```

Sorting

- Sorting is the process of arranging a list of items in certain order, e.g. numerical order or lexicographical order.
- Many applications require sorting:
  - To order a group of students according to their names, ID, and examinations scores.
  - To arrange a list of events in chronological order.
  - To facilitate the search for information, e.g. dictionary or phone books.
  - To display a list of webpages based on their popularity, e.g. number of hits.

Selection Sort

- Selection sort performs sorting by repeatedly finding the largest element in the unsorted portion of the array and then placing it to the end of this unsorted portion until the whole array is sorted.
- **Algorithm**
  - Define the entire array as unsorted at the beginning
  - While the unsorted portion of the array has more than one element:
    - Find its largest element
    - Swap with last element
    - Reduce the unsorted portion of the array by 1

Selection Sort - Microsoft Internet Explorer

Array

3	10	4	6	8	9	7	2	1	5

Description

Search through the array, find the largest element (10), and swap it with the last element (5), in the unprocessed portion of the array since they are different.

Note that the whole array is unprocessed at the beginning (which is in yellow).

Algorithm

1. Define the "unprocessed" portion of the array.

2. **While the unprocessed portion of the array has more than one element:**

   2.1 Find largest element

   2.2 Swap with last element if they are different

   2.3 Reduce unprocessed portion of the array by 1

Change Array | Progra

Selection Sort

```
/*
 * Use selection sort to arrange the array in ascending order
 */
public void selectSort () {
    int maxPos;  // index for the largest element in unsorted array

    for (int i = scoreArray.length-1; i > 0; i--) {
        maxPos = maxIndex(i+1);        // find the largest element
        swap (scoreArray, maxPos, i); // swap the largest and last
                                       // elements of unsorted portion
    }
}
```

Using Break and Continue

- Two statements: **break** and **continue** can be used in all 3 types of loops
- Usage of break statement
  - Conditionally terminate and exit the loop
- Usage of continue statement
  - Conditionally skip the remaining statements in the loop body and start the next iteration

Example

```
int[ ] intArray = {90,78,100,90,65};
int value = 100;  // value to search for
int size = intArray.length;
boolean found = false;
int i;
for (i = 0; i < size; i++) {
    if (intArray[i] == value) {
        found = true;
        break;
    }
}
if (found)
    IO.outputln("The value was found at index" + i);
else
    IO.outputln{“The value was not found”);
```

Example

```
int[ ] intArray = {90,78,100,90,65};
int value = 90;  // value to search for
int size = intArray.length;
int nTimes = 0;
int i;
for (i = 0; i < size; i++) {
    if (intArray[i] != value) continue;
    // actions for each occurrence of value
    nTimes++;
}

IO.outputln("The value was found " + nTimes + " times.");
```

Two-dimensional Array

- The idea of one-dimensional array can be extended to two-dimensional
- A R x C two dimensional array can be illustrated as a table with R rows and C columns
- Example: The following scores could be stored for each student in a course:
  - Exam score
  - Homework score
  - Lab score
  - Final score

Two-dimensional Array

FIFA.com 2014 FIFA World Cup Brazil™

Groups

LEARN MORE

Group A | Group B | Group C | Group D | Group E | Group F | Group G | Group H | Qualifiers | Share

GROUP G

TEAMS	MP	W	D	L	GF	GA	Pts
GERMANY	3	2	1	0	7	2	7
USA	3	1	1	1	4	4	4
PORTUGAL	3	1	1	1	4	7	4
GHANA	3	0	1	2	4	6	1

Group Details ›

Example: Students' test scores

double[ ][ ] scores;

Index

0	99.0
1	90.0
2	85.0
3	72.0

double[ ] testScore

scores (column index)

Student numbers (row index)

	Test1 0	Test2 1	Test3 2	Test4 3
0	99.0	89.0	85.0	92.0
1	90.0	74.0	75.0	82.0
2	85.0	75.0	64.0	91.0
3	72.0	82.0	81.0	94.0

Example: Scores

```
public class Scores {
    /* 1. A 2D array instance variable */

    /* 2. Initialize a 2D array */

    /* 3. Access a 2D array element */

    /* 4. Traverse a 2D array
                using a nested loop */

}
```

Example: Declare Scores

```
public class Scores {
/* 1. A 2D array instance variable */
   private double [ ][ ] scores ;

   /* 2. Initialize a 2D array */

   /* 3. Access a 2D array element */

   /* 4. Traverse a 2D array
              using a nested loop */

}
```

Define an instance variable of a 2D array
Syntax:
DataType[ ][ ] nameOfTheVariable;

Initializing Scores

```
public class Scores {
    /* 1. A 2D array instance variable */
    private double [ ][ ] scores ;

    public void initializeAllScores( ) {

        /* 2. Initialize a 2D array */

    }

    /* 3. Access a 2D array element */
    /* 4. Traverse a 2D array using a nested loop */
}
```

Initializing Scores

```
public void initializeAllScores( ) {

 scores = new double[4][4];

}
```

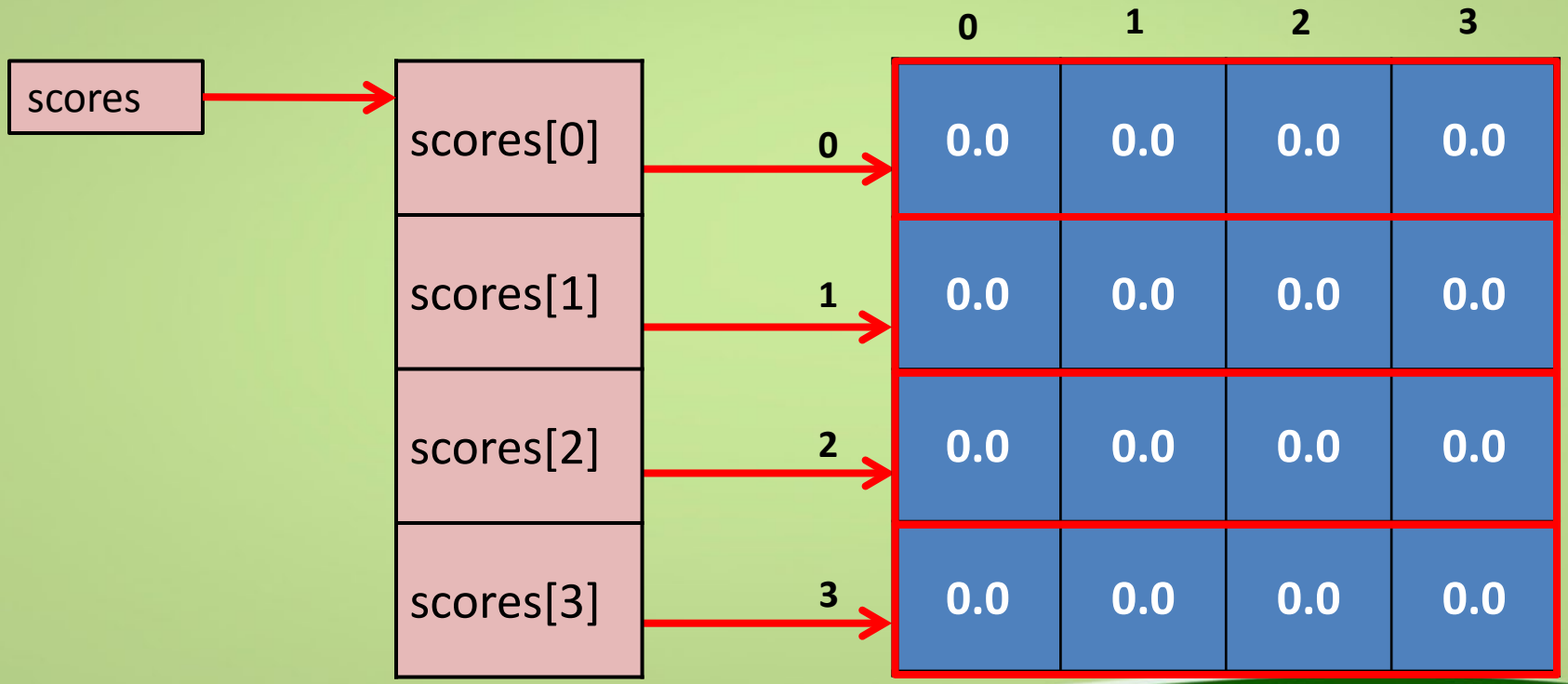
double[ ][ ] scores

	0	1	2	3
0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0

An array of 4 rows and 4 columns is created

2D array as an Array of 1D arrays

- Each row can be visualized as a 1D array
  - **double[ ][ ] scores = new double[4][4];**



Initializing Scores

```
public void initializeAllScores() {
scores = new double[4][4];

scores[0][0] = 99.0; scores[0][1] = 89.0;
scores[0][2] = 85.0; scores[0][3] = 92.0;

}
```

The first index is the row index and the second is the column index. Both indices start from 0

double[ ][ ] scores

	0	1	2	3
0	99.0	89.0	85.0	92.0
1	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0

Initializing Scores

```
public void initializeAllScores() {
    scores = new double[4][4];

      scores[0][0] = 99.0; scores[0][1] = 89.0;
      scores[0][2] = 85.0; scores[0][3] = 92.0;

      scores[1][0] = 90.0; scores[1][1] = 74.0;
      scores[1][2] = 75.0; scores[1][3] = 82.0;
}
```

Initializing the second row

double[ ][ ] fares

	0	1	2	3
0	99.0	89.0	85.0	92.0
1	90.0	74.0	75.0	82.0
2	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0

Initializing Scores

```
public void initializeAllScores() {
 scores = new double[4][4];

  scores[0][0] = 99.0; scores[0][1] = 89.0;
  scores[0][2] = 85.0; scores[0][3] = 92.0;
  scores[1][0] = 90.0; scores[1][1] = 74.0;
  scores[1][2] = 75.0; scores[1][3] = 82.0;
  scores[2][0] = 85.0; scores[2][1] = 75.0;
  scores[2][2] = 64.0; scores[2][3] = 91.0;

 }
```

Initializing the third row

double[ ][ ] scores

	0	1	2	3
0	99.0	89.0	85.0	92.0
1	90.0	74.0	75.0	82.0
2	85.0	75.0	64.0	91.0
3	0.0	0.0	0.0	0.0

Initializing Scores

```
public void initializeAllScores() {
 scores = new double[4][4];
  scores[0][0] = 99.0; scores[0][1] = 89.0;
  scores[0][2] = 85.0; scores[0][3] = 92.0;
  scores[1][0] = 90.0; scores[1][1] = 74.0;
  scores[1][2] = 75.0; scores[1][3] = 82.0;
  scores[2][0] = 85.0; scores[2][1] = 75.0;
  scores[2][2] = 64.0; scores[2][3] = 91.0;

  scores[3][0] = 72.0; scores[3][1] = 82.0;
  scores[3][2] = 81.0; scores[3][3] = 94.0;

}
```

Initializing the forth row

double[ ][ ] scores

	0	1	2	3
0	99.0	89.0	85.0	92.0
1	90.0	74.0	75.0	82.0
2	85.0	75.0	64.0	91.0
3	72.0	82.0	81.0	94.0

Shorthand notation for a 2D array

- Declare, define and initialize a 2D array using a single statement:

```
double [ ][ ] scores = {
    {99.0, 89.0, 85.0, 92.0},
    {90.0, 74.0, 75.0, 82.0},
    {85.0, 75.0, 64.0, 91.0},
    {72.0, 82.0, 81.0, 94.0}
};
```

	0	1	2	3
0	99.0	89.0	85.0	92.0
1	90.0	74.0	75.0	82.0
2	85.0	75.0	64.0	91.0
3	72.0	82.0	81.0	94.0

Caution on the shorthand syntax

- This shorthand syntax **must be in one statement**

```
double [ ][ ] scores = {
    {99.0, 89.0, 85.0, 92.0},
    {90.0, 74.0, 75.0, 82.0},
    {85.0, 75.0, 64.0, 91.0},
    {72.0, 82.0, 81.0, 94.0}
};
```

```
double [ ][ ] scores;
scores = {
    {99.0, 89.0, 85.0, 92.0},
    {90.0, 74.0, 75.0, 82.0},
    {85.0, 75.0, 64.0, 91.0},
    {72.0, 82.0, 81.0, 94.0}
};
```

Access a 2D array element

```
public class Scores{

    /* 1. A 2D array instance variable */
    private double[ ][ ] scores;
    public void initializeAllScores() {   /* 2. Initialize a 2D array */    }

    public double getScoreByIndices(int rowIndex, int colIndex) {
        /* 3. Access a 2D array element */
    }

    /* 4. Traverse a 2D array
                using a nested loop */

}
```

Access an element in a 2D array

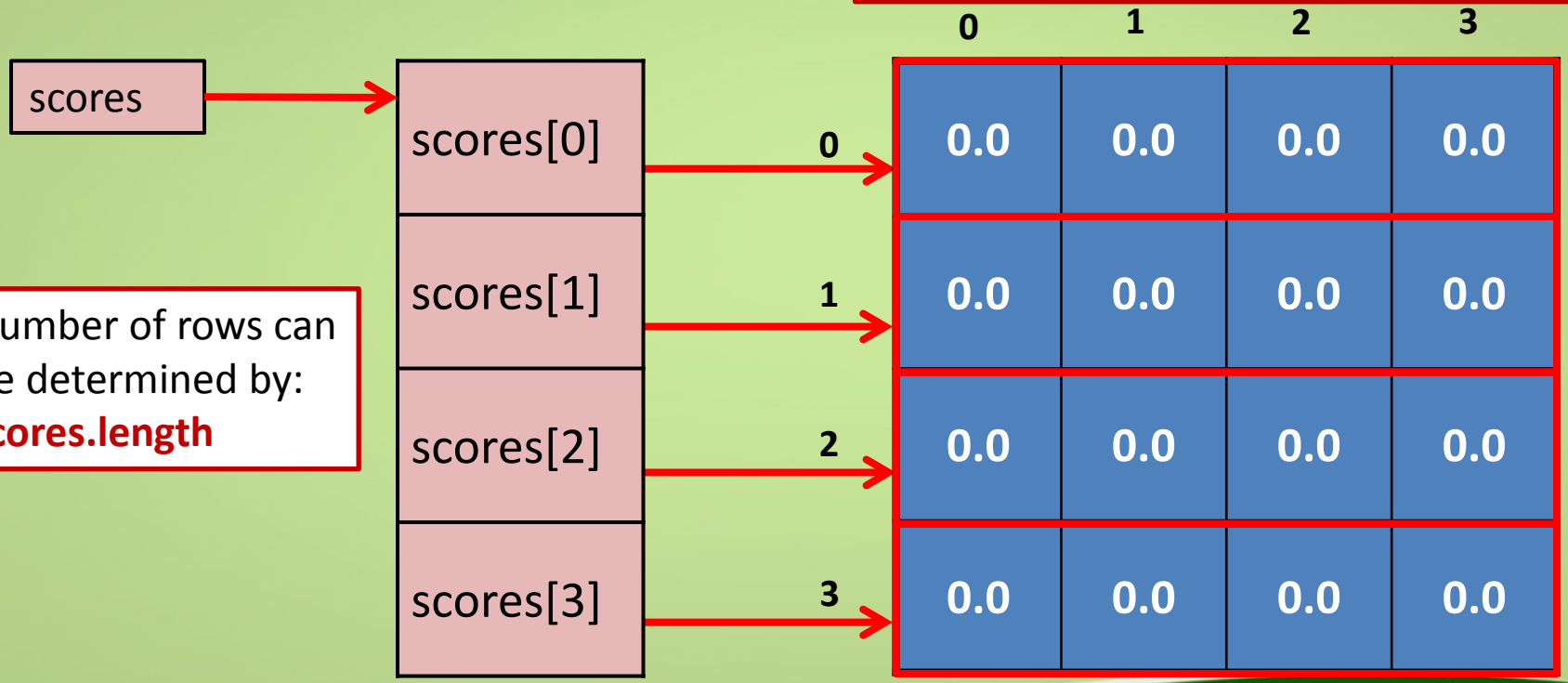
```
public double getScoreByIndices(int rowIndex, int colIndex)
{

    int numOfRows = scores.length;
    int numOfCols = scores[0].length;

}
```

Get the number of rows and the number of columns
In this example, we assume that all rows have the same number of elements

2D array as an Array of 1D arrays

- Each row can be visualized as a 1D array
  - **double[ ][ ] scores = new double[4][4];**

Number of columns can be determined by:
scores[i].length



Number of rows can be determined by:
scores.length

Access an element in a 2D array

```
public double getScoreByIndices(int rowIndex, int colIndex)
{

    int numOfRows = scores.length;
    int numOfCols = scores[0].length;

}
```

Get the number of rows and the number of columns
In this example, we assume that all rows have the same number of elements

Access an element in a 2D array

```
public double getScoreByIndices(int rowIndex, int colIndex) {
        int numOfRows = scores.length;
        int numOfCols = scores[0].length;

        if ( rowIndex < 0 || rowIndex >= numOfRows )
            return -1.0;
        if ( colIndex < 0 || colIndex >= numOfCols )
            return -1.0;

}
```

If the row index or the column index is invalid, return -1.0

Access an element in a 2D array

```
public double getScoreByIndices(int rowIndex, int colIndex) {
        int numOfRows = scores.length;
        int numOfCols = scores[0].length;

        if ( rowIndex < 0 || rowIndex >= numOfRows )
            return -1.0;
        if ( colIndex < 0 || colIndex >= numOfCols )
            return -1.0;

        return scores[rowIndex][colIndex];
}
```

Return the score located by the rowIndex (the first index) and the colIndex (the second index)

Traverse a 2D Array using a Nested Loop

```
public class Scores {
   /* 1. A 2D array instance variable */
   private double[ ][ ] scores;
   public void initializeAllScores() {   /* 2. Initialize a 2D array */   }
   public double getScoreByIndices(int rowIndex, int colIndex) {
         /* 3. Access a 2D array element */
    }

   public void printAllScores() {
         /* 4. Traverse a 2D array using a nested loop */
   }

}
```

Traversing a 2D array using a nested loop

```
public void printAllScores() {

 int numOfRows = scores.length;
 int numOfCols = scores[0].length;

}
```

Get the number of rows and the number of columns. In this example, both are 4.

double[ ][ ] fares

	0	1	2	3
0	99.0	89.0	85.0	92.0
1	90.0	74.0	75.0	82.0
2	85.0	75.0	64.0	91.0
3	72.0	82.0	81.0	94.0

Traversing a 2D array using a nested loop

```
public void printAllScores() {
 int numOfRows = scores.length;
 int numOfCols = scores[0].length;
 for ( int r=0; r<numOfRows; r++) {
   IO.output("Row " + r + " : ");
   for (int c=0; c<numOfCols; c++) {
      IO.output(getScoreByIndices(r,c) + " " );
    } // for loop c
    IO.outputln(" " );
 }  // for loop r
}   // end of the method
```

r = 0

c = 3

double[ ][ ] scores

	0	1	2	3
0	99.0	89.0	85.0	92.0
1	90.0	74.0	75.0	82.0
2	85.0	75.0	64.0	91.0
3	72.0	82.0	81.0	94.0

Traversing a 2D array using a nested loop

```
public void printAllScores() {
 int numOfRows = scores.length;
 int numOfCols = scores[0].length;
 for ( int r=0; r<numOfRows; r++) {
   IO.output("Row " + r + " : ");
   for (int c=0; c<numOfCols; c++) {
      IO.output(getScoreByIndices(r,c) + " " );
    } // for loop c
    IO.outputln(" " );
 }  // for loop r
}   // end of the method
```

r = 1

c = 3

double[ ][ ] scores

	0	1	2	3
0	99.0	89.0	85.0	92.0
1	90.0	74.0	75.0	82.0
2	85.0	75.0	64.0	91.0
3	72.0	82.0	81.0	94.0

Traversing a 2D array using a nested loop

```
public void printAllScores() {
 int numOfRows = scores.length;
 int numOfCols = scores[0].length;
 for ( int r=0; r<numOfRows; r++) {
   IO.output("Row " + r + " : ");
   for (int c=0; c<numOfCols; c++) {
      IO.output(getScoreByIndices(r,c) + " " );
    } // for loop c
    IO.outputln(" " );
 }  // for loop r
}   // end of the method
```

r = 3

c = 3

double[ ][ ] scores

	0	1	2	3
0	99.0	89.0	85.0	92.0
1	90.0	74.0	75.0	82.0
2	85.0	75.0	64.0	91.0
3	72.0	82.0	81.0	94.0

Compute Average

```
/*
 * aveScore computes the average of the values in an array
 */
public double aveScore( ) {
    double sum = 0;      // for storing the cumulative sum
    int size = scoreArray.length;  // size of the array

    for (int i = 0; i < size; i++)
        sum = sum + scoreArray[i];

    return sum / size;
  }
```

Example: Students' test scores

<u>double[ ][ ] scores;</u>

scores (column index)

Student numbers (row index)	Test1 0	Test2 1	Test3 2	Test4 3
0	99.0	89.0	85.0	92.0
1	90.0	74.0	75.0	82.0
2	85.0	75.0	64.0	91.0
3	72.0	82.0	81.0	94.0

Compute Average by Row

```
/*
 * aveByRow computes the row average of an array
 */
public double aveByRow( int row ) {
    double sum = 0;      // for storing the cumulative sum
    int numOfCols = scores[row].length;

    for (int c = 0; c < numOfCols; c++)
      sum = sum + scores[row][c];

    return sum / numOfCols;
}
```

Compute Average by Column

```
/*
 * aveByCol computes the column average an array
 */
public double aveByCol( int col ) {
    double sum = 0;      // for storing the cumulative sum
    int numOfRows = scores.length;

    for (int r = 0; r < numOfRows; r++)
      sum = sum + scores[r][col];

    return sum / numOfRowls;
}
```

Find Maximum

```
/*
 * maxIndex finds the location of the largest values in an array
 * up to index size - 1
 */
public int maxIndex(int size){
    int mIndex = 0;   // index for the current maximum
    if (size > scoreArray.length) size = scoreArray.length;

    for (int i = 0; i < size; i++) {
        if (scoreArray[i] > scoreArray[mIndex]) mIndex = i;
    }
    return mIndex;
}
```

Find Maximum 2D

```
/*
 * maxRowIndex finds the location of the largest values for a
 * given column in a 2D array
 */
public int maxRowIndex(int col, int size){
    int mIndex = 0;   // index for the current maximum
    if (size > scores.length) size = scores.length;

    for (int i = 0; i < size; i++) {
        if (scores[i][col] > scores[mIndex][col]) mIndex = i;
    }
    return mIndex;
}
```