

Arrays of Objects

zyBook Chap 7.14

Arrays of Objects

- This lecture covers...
 - Array of Strings
 - Array of Arrays

Array of Strings

We've been seeing a String array since the HelloWorld program...

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

It is **NOT** necessary to name the String array as **args**, just a convention

String[] args – Command-Line Arguments

```
public class ArgsDemo {  
    public static void main(String[] args) {  
        System.out.println("Received " + args.length + " arguments.");  
  
        for(int i = 0; i < args.length; ++i){  
            System.out.println("args[" + i + "]: " + args[i]);  
        }  
    }  
}
```

```
$ javac ArgsDemo.java
```

```
$ java ArgsDemo
```

```
Received 0 arguments.
```

```
$ java ArgsDemo Hello World
```

```
Received 2 arguments.
```

```
args[0]: Hello
```

```
args[1]: World
```

```

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class CountTokensInFile {
    public static void main (String[] args) throws FileNotFoundException {

        // Set up the scanner to read in from file
        String fileName = "CourseDescription.txt";
        File fileInput = new File(fileName);
        Scanner input = new Scanner(fileInput);

        int countToken = 0, countInt = 0;

        while (input.hasNext()) {
            if (input.hasNextInt()) {
                ++countInt;
            }
            input.next();
            ++countToken;
        }
        input.close(); // Close the scanner
        System.out.println(fileName + " has " + countToken + " tokens, including " +
                           countInt + " integer(s).");
    }
}

```

Lec21 Example: Token-Based File Processing

```

$ javac CountTokensInFile.java
$ java CountTokensInFile
CourseDescription.txt has 80 tokens, including 4 integer(s).

```

```

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class CountTokensInFile {
    public static void main (String[] args) throws FileNotFoundException {

        // Scanner to read from console
        Scanner console = new Scanner(System.in);
        System.out.print("Enter file name: ");
        String fileName = console.next();

        // Scanner to read from file
        File fileInput = new File(fileName);
        Scanner input = new Scanner(fileInput);

        int countToken = 0, countInt = 0;

        while (input.hasNext()) {
            if (input.hasNextInt()) {
                ++countInt;
            }
            input.next();
            ++countToken;
        }
        input.close(); // Close the scanner for file
        console.close(); // Close the scanner for console for better practice

        System.out.println(fileName + " has " + countToken +
            " tokens, including " + countInt + " integer(s).");
    }
}

```

Using Console Input for File Name

```

$ javac CountTokensInFile.java
$ java CountTokensInFile
Enter file name: CourseDescription.txt
CourseDescription.txt has 80 tokens, including 4 integer(s).

```

```

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class CountTokensInFile {
    public static void main (String[] args) throws FileNotFoundException {
        // Check if the user gave an argument for file name
        if (args.length != 1) {
            System.out.println("Usage: java CountTokensInFile input_filename");
            System.exit(1);
        }
        String fileName = args[0];

        File fileInput = new File(fileName);
        Scanner input = new Scanner(fileInput);

        int countToken = 0, countInt = 0;

        while (input.hasNext()) {
            if (input.hasNextInt()) {
                ++countInt;
            }
            input.next();
            ++countToken;
        }
        input.close(); // Close the scanner

        System.out.println(fileName + " has " + countToken +
                           " tokens, including " + countInt + " integer(s).");
    }
}

```

Using Command Line Arguments for File Name

```

$ javac CountTokensInFile.java
$ java CountTokensInFile
Usage: java CountTokensInFile input_filename
$ java CountTokensInFile CourseDescription.txt
CourseDescription.txt has 80 tokens, including 4 integer(s).

```

Default Value of an Object

Q: What's the output of the following code?

```
import java.util.Arrays;

public class CountLetter {
    public static void main(String[] args) {
        String[] words = new String[5];
        System.out.println(Arrays.toString(words));

        // more code here...
    }
}
```

[null, null, null, null, null]

null – Default Value of an Object

- A value that shows that the object is referring to nothing

It is legal to...

- **store** null in a variable or array element
 - often as an initial value to be overwritten later
- **print** a null reference
- **ask whether** a variable or array element is null
- **pass** null as a parameter to a method
- **return** null from a method
 - often as an indication of failure, such as a method that searches for an object in a file/array but does not find it

NullPointerException

It is NOT legal to **dereference** an object that is `null`

- That is, try to **access** any of **its methods or data** using `.` (dot) notation

Two-Phase Initialization

- Arrays of objects should use a 2-phase initialization

```
public class CountLetter {  
    public static void main(String[] args) {  
  
        // Phase 1: Initializing the array itself  
        String[] words = new String[5];  
  
        // Phase 2: Initializing the object stored into each element of the array  
        words[0] = "Hello";  
        words[2] = "World";  
  
        // more code here...  
    }  
}
```

Q: What's wrong with the following code?

```
/**
 * This program counts the number of letters in each String elements in an array.
 */
public class CountLetter {
    public static void main(String[] args) {
        String[] words = new String[5];
        words[0] = "Hello";
        words[2] = "World";

        // words[1], words[3], and words[4] are null
        int numLetter = 0;
        for (int i = 0; i < words.length; ++i) {
            System.out.println("words[" + i + "]: " +
                               words[i].length() + " letters");
        }
    }
}
```

```
$ javac CountLetter.java
```

```
$ java CountLetter
```

```
words[0]: 5 letters
```

```
Exception in thread "main" java.lang.NullPointerException:
Cannot invoke "String.length()" because "<local1>[<local3>]" is null
at CountLetter.main(CountLetter.java:14)
```

Avoiding NullPointerException

```
/**
 * This program counts the number of letters in each String elements in an array.
 */
public class CountLetter {
    public static void main(String[] args) {
        String[] words = new String[5];
        words[0] = "Hello";
        words[2] = "World";

        // words[1], words[3], and words[4] are null
        int numLetter = 0;
        for (int i = 0; i < words.length; ++i) {
            if (words[i] != null) {
                System.out.println("words[" + i + "]: " +
                                   words[i].length() + " letters");
            }
        }
    }
}
```

When dealing with elements that may be **null**, you ALWAYS check for **null** before any calls

```
$ javac CountLetter.java
$ java CountLetter
words[0]: 5 letters
words[2]: 5 letters
```

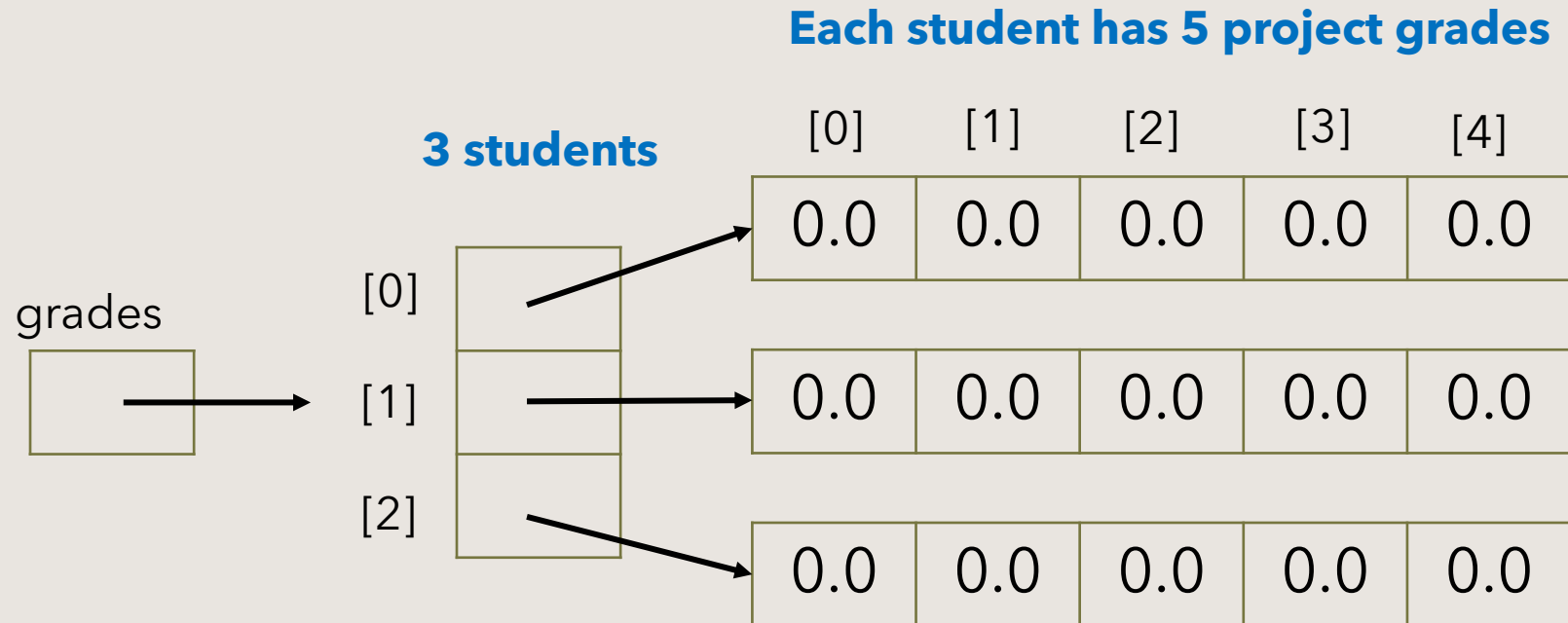
Array of Arrays (Multi-Dimensional Arrays)

- `int` → one integer
- `int[]` → a **one**-dimensional array of integers
- `int[][]` → An array of int arrays
 - **two**-dimensional grid of integers
 - Convention: `[<rows>][<columns>]`
- `int[][][]` ...

Example

2-D array storing grades for **three** students, where each student has **five** project grades

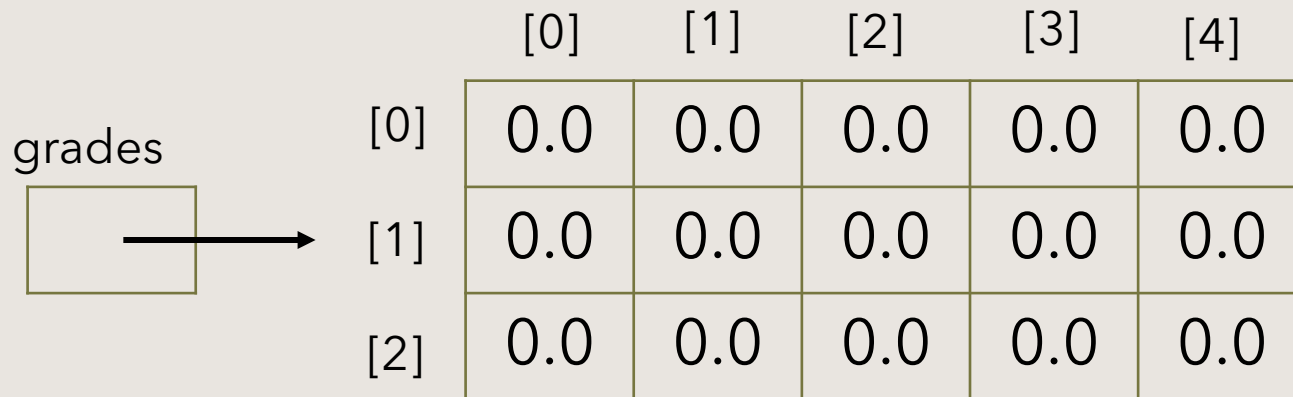
- `double[][] grades = new double[3][5];`



Rectangular/Grid Representation for 2D Array

2-D array storing grades for **three** students, where each student has **five** project grades

- `double[][] grades = new double[3][5];`

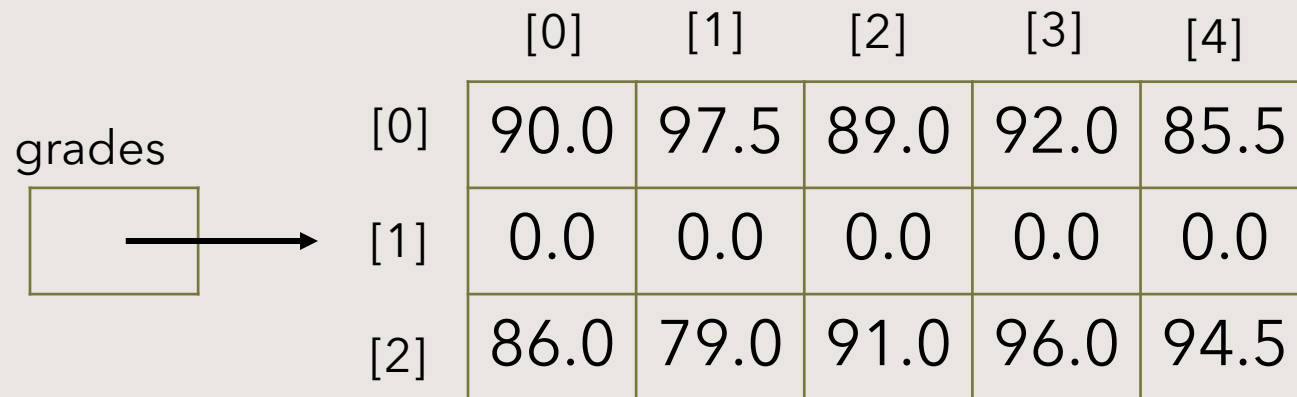


The diagram illustrates the memory representation of the 2D array. A variable named 'grades' is shown with a small rectangular box next to it. An arrow points from this box to a larger grid representing the array. The grid has 3 rows and 5 columns. The rows are indexed [0], [1], and [2] on the left. The columns are indexed [0], [1], [2], [3], and [4] at the top. Each cell in the grid contains the value 0.0.

	[0]	[1]	[2]	[3]	[4]
[0]	0.0	0.0	0.0	0.0	0.0
[1]	0.0	0.0	0.0	0.0	0.0
[2]	0.0	0.0	0.0	0.0	0.0

Access 2D Array Elements

- `grades` → the entire 2-d array
- `grades[1]` → the entire row 1 // **[0.0, 0.0, 0.0, 0.0, 0.0]**
- `grades[0][0]` → element at row 0, column 0 // **90.0**
- `grades[2][3]` // **96.0**
- `grades[2][4]` // **94.5**

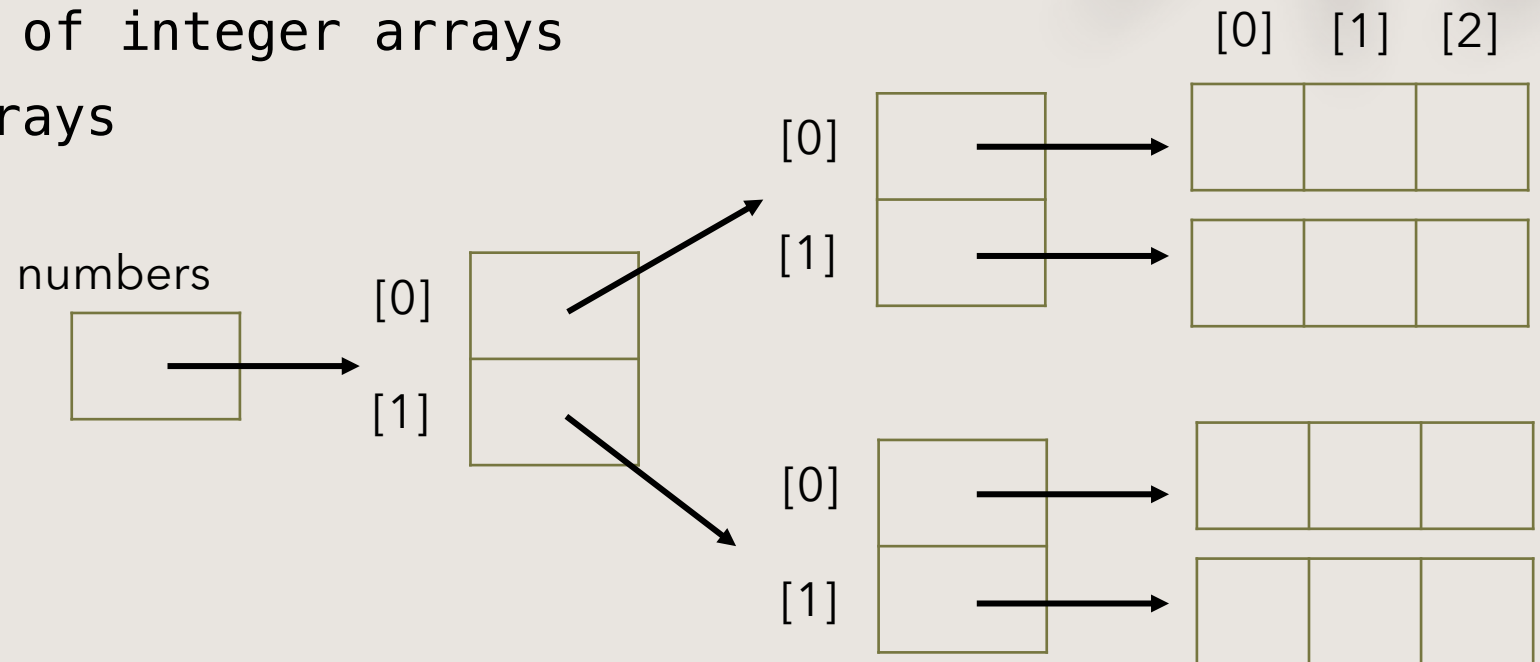


The diagram illustrates the variable 'grades' pointing to a 2D array. A box labeled 'grades' has an arrow pointing to the first column of a table. The table has 3 rows and 5 columns, with row indices [0], [1], [2] and column indices [0], [1], [2], [3], [4].

	[0]	[1]	[2]	[3]	[4]
[0]	90.0	97.5	89.0	92.0	85.5
[1]	0.0	0.0	0.0	0.0	0.0
[2]	86.0	79.0	91.0	96.0	94.5

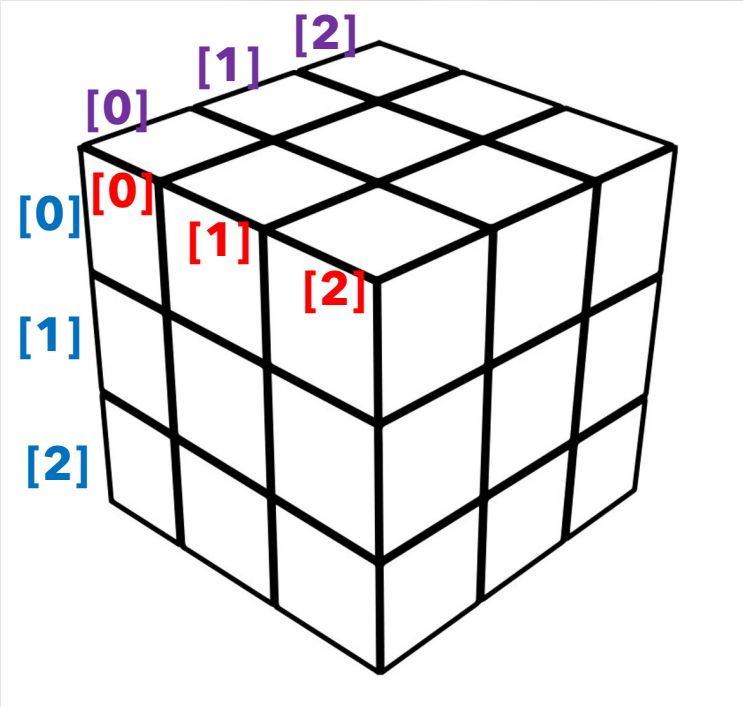
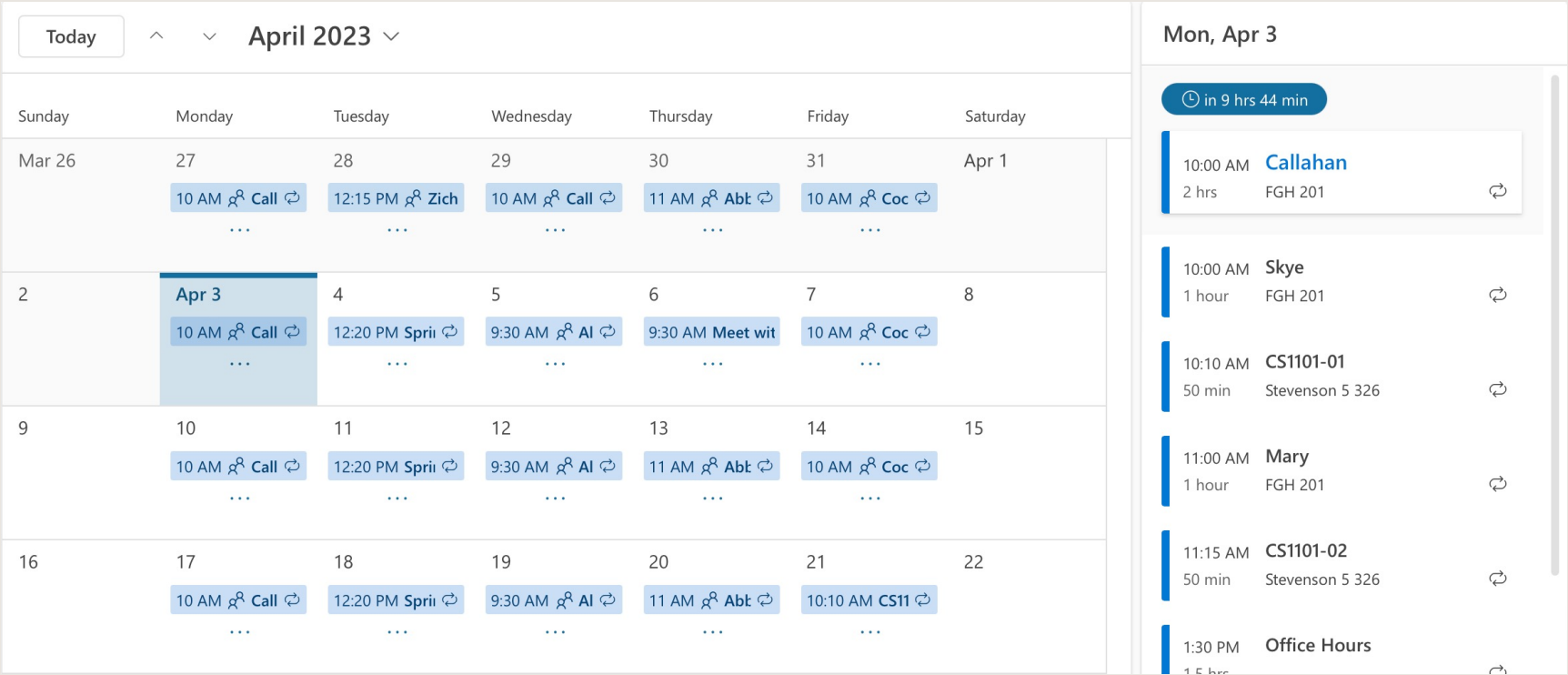
Multi-dimensional Arrays

- Three-dimensional arrays
 - For example
 - `int[][][] numbers = new int[2][2][3];`
 - Array of arrays of integer arrays
 - Array of 2-d arrays



Daily Life 3D Array Examples

1st Dimension: 4 Weeks
2nd Dimension: 7 Days in each week
3rd Dimension: X Tasks in each day



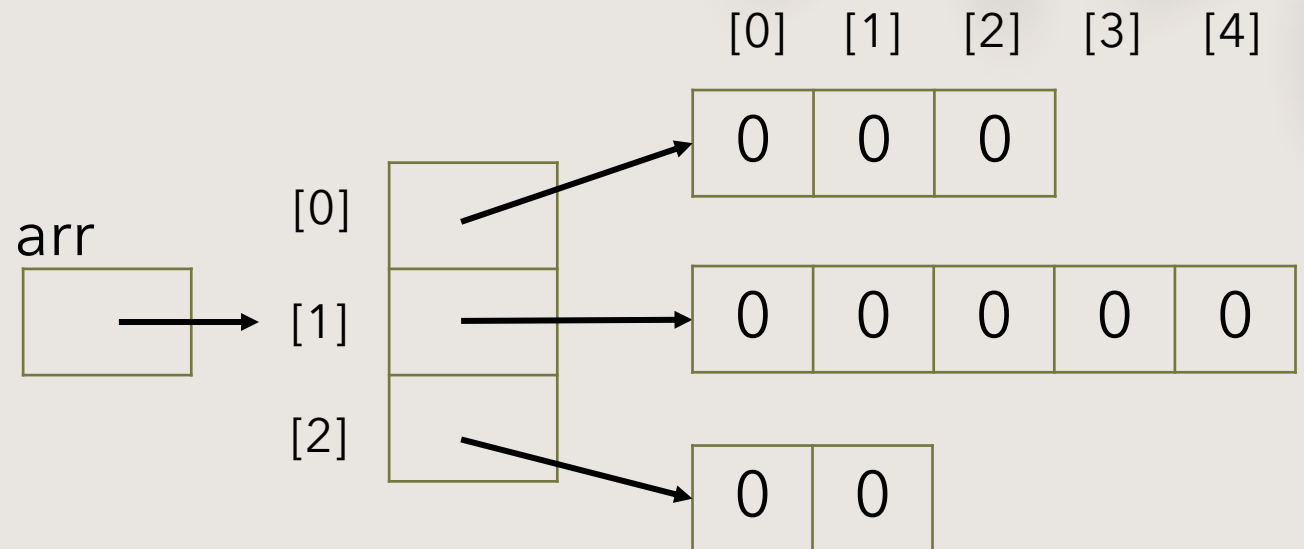
Generalizing Multi-dimensional Arrays

- Multi-dimensional arrays
 - Consistency on what you consider each array of arrays to be
 - Comments to remind you (and others) what each dimension is—program context!

Jagged Arrays

- An array of arrays of varying lengths
 - First, construct the first-dimension array (the "rows").
Then, construct the array for each row.

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



Coding Practice

MultiplicationTable



JAVA File



Complete the **MultiplicationTable** program that contains methods that create multiplication tables with a given number of rows and columns and prints them.

```
public class MultiplicationTable {  
  
    public static void main(String[] args) {  
        System.out.println();  
        int[][] smallTable = createMultiplicationTable(3);  
        printMultiplicationTable(smallTable);  
  
        System.out.println();  
        int[][] largeTable = createMultiplicationTable(10);  
        printMultiplicationTable(largeTable);  
    }  
  
    //Create and return a multiplication table with n rows and n columns  
    public static int[][] createMultiplicationTable(int n) {  
        // TODO: Add code here  
    }  
  
    //Print multiplication table using %4d to print each value  
    public static void printMultiplicationTable(int[][] table) {  
        // TODO: Add code here  
    }  
}
```

\$ java MultiplicationTable

1	2	3
2	4	6
3	6	9

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Sample Solution

```
public class MultiplicationTable {
    public static void main(String[] args) {
        System.out.println();
        int[][] smallTable = createMultiplicationTable(3);
        printMultiplicationTable(smallTable);

        System.out.println();
        int[][] largeTable = createMultiplicationTable(10);
        printMultiplicationTable(largeTable);
    }

    public static int[][] createMultiplicationTable(int n) {
        int[][] multTable = new int[n][n];
        for (int row = 0; row < multTable.length; ++row) {
            for (int col = 0; col < multTable[row].length; ++col) {
                multTable[row][col] = (row + 1) * (col + 1);
            }
        }

        return multTable;
    }

    public static void printMultiplicationTable(int[][] table) {
        for (int row = 0; row < table.length; ++row) {
            for (int col = 0; col < table[row].length; ++col) {
                System.out.printf("%4d", table[row][col]);
            }
            System.out.println("");
        }
    }
}
```