2D Arrays

CS 1400 2021

Cody Henrichsen

2D Structure

- Data organized into a 2D grid or matrix
- In Java it is really an array of arrays
- Two [ ] are the “keyword” since an array is hipster
- T/type [ ][ ] variableName = new T/type [rows][cols];
- Rows are always the first set of [] columns the second!

2D Addresses

arrayName [0][0]	arrayName [0][1]	arrayName [0][2]	arrayName [0][3]	arrayName [0][4]
arrayName [1][0]	arrayName [1][1]	arrayName [1][2]	arrayName [1][3]	arrayName [1][4]
arrayName [2][0]	arrayName [2][1]	arrayName [2][2]	arrayName [2][3]	arrayName [2][4]
arrayName [3][0]	arrayName [3][1]	arrayName [3][2]	arrayName [3][3]	arrayName [3][4]

2D Info

- Just like an array the placement of the assignment operator tells you what is happening
  - Right of the arrayName and address is assigning a value to the location
  - Left of the arrayName and address is retrieving a value from the array

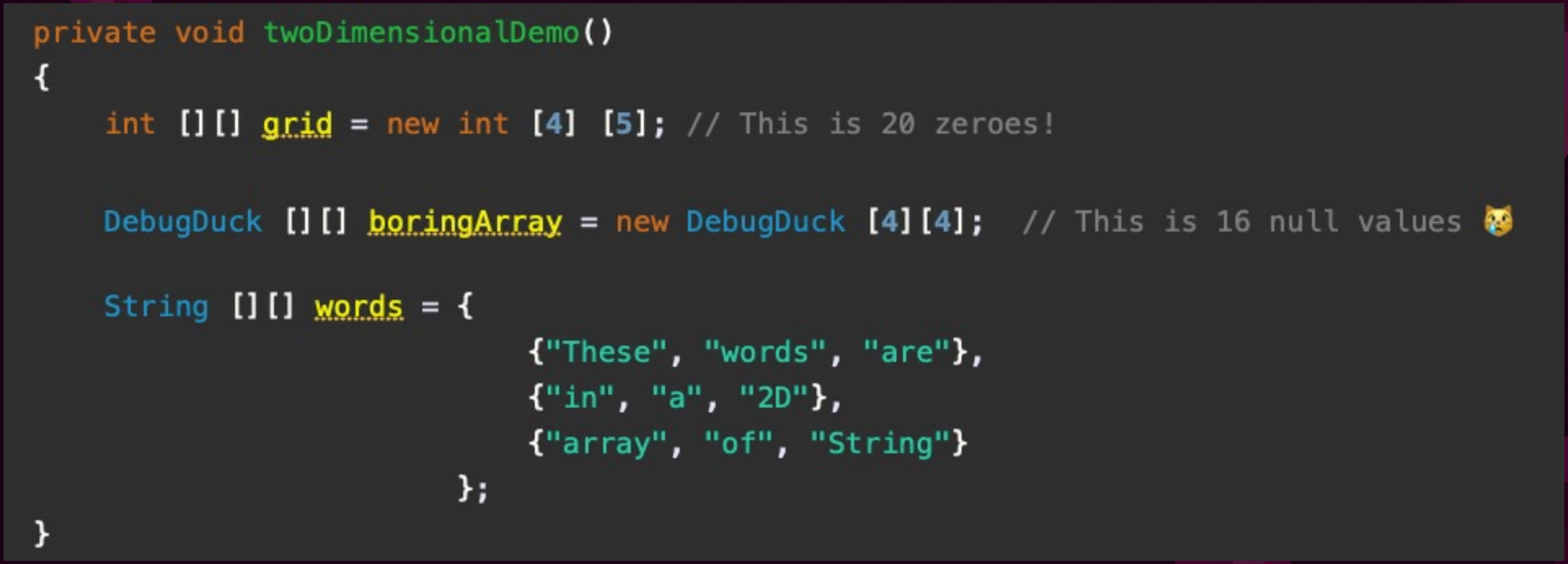
Columns

[0].length = Columns

2D Initialization

- The default value for the data type of the array fills each slot
  - 0 for primitives except boolean (false)
  - null for all Objects
- Can also initialize with values separated by commas surrounded with squiggles for each row in the array

2D Initialization Demo

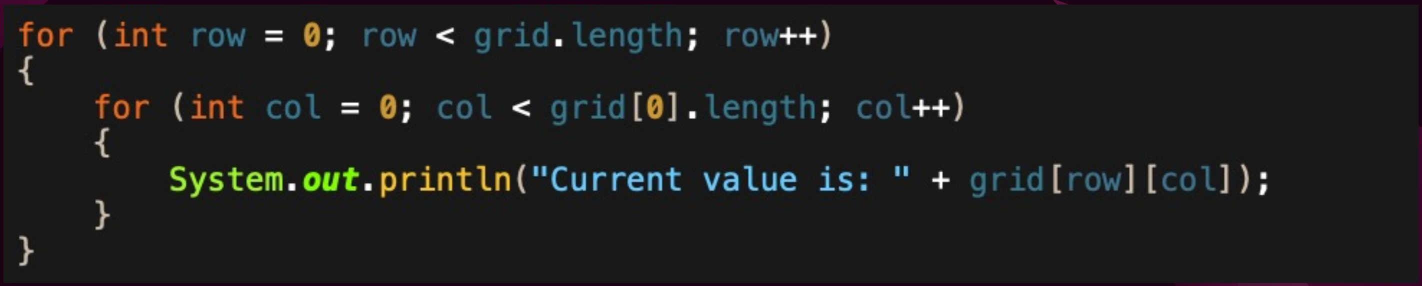


```java
private void twoDimensionalDemo()
{
    int [][] grid = new int [4] [5]; // This is 20 zeroes!

    DebugDuck [][] boringArray = new DebugDuck [4][4];  // This is 16 null values 😿

    String [][] words = {
                            {"These", "words", "are"},
                            {"in", "a", "2D"},
                            {"array", "of", "String"}
                        };
}
```

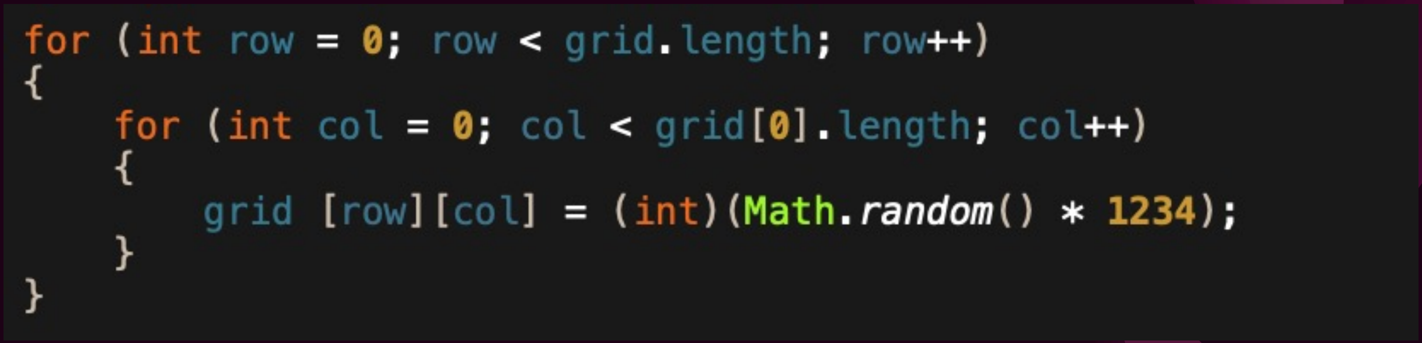
Iteration: Standard For

- Proper iteration of a 2D array can be either row major or column major
  - AKA loop over rows first or columns first
- Regardless of style you will want to nest the loops
- If you need addresses you will need the standard for loop
  - Replacing data AKA assigning values
- You can iterate over parts of the 2D array and vary the steps
  - Backwards
  - Half
  - Every third/seventh spot…

Iteration – Standard For Demo



```java
for (int row = 0; row < grid.length; row++)
{
    for (int col = 0; col < grid[0].length; col++)
    {
        System.out.println("Current value is: " + grid[row][col]);
    }
}
```

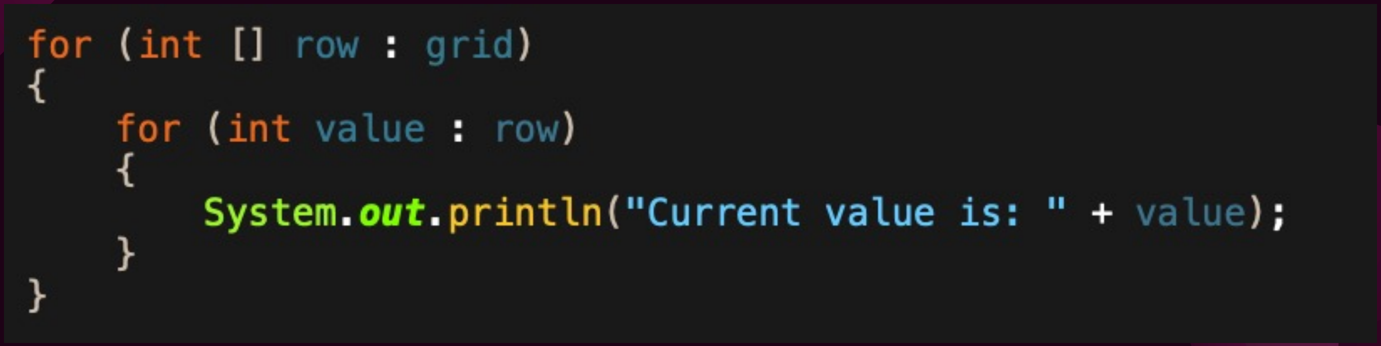


```java
for (int row = 0; row < grid.length; row++)
{
    for (int col = 0; col < grid[0].length; col++)
    {
        grid [row][col] = (int)(Math.random() * 1234);
    }
}
```

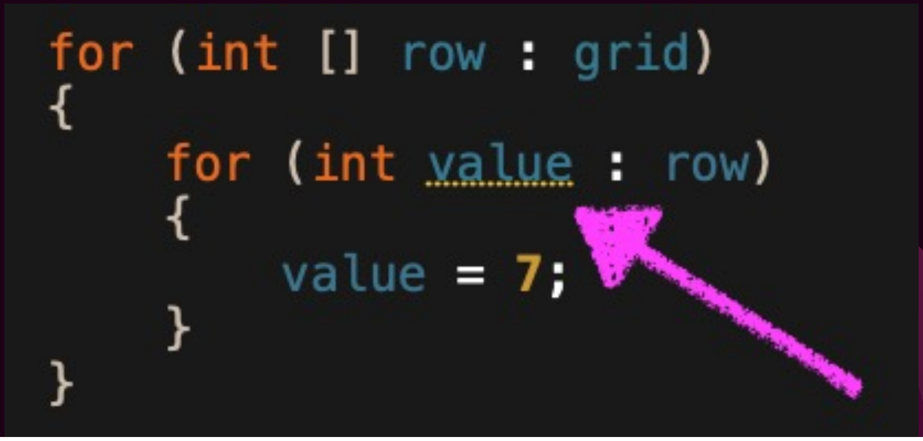
Iteration: For Each

- Great for output or evaluating values
- Can’t replace values in a for-each!
- Objects can have mutator methods called to update state!
- Row major only

Iteration – For Each Demo

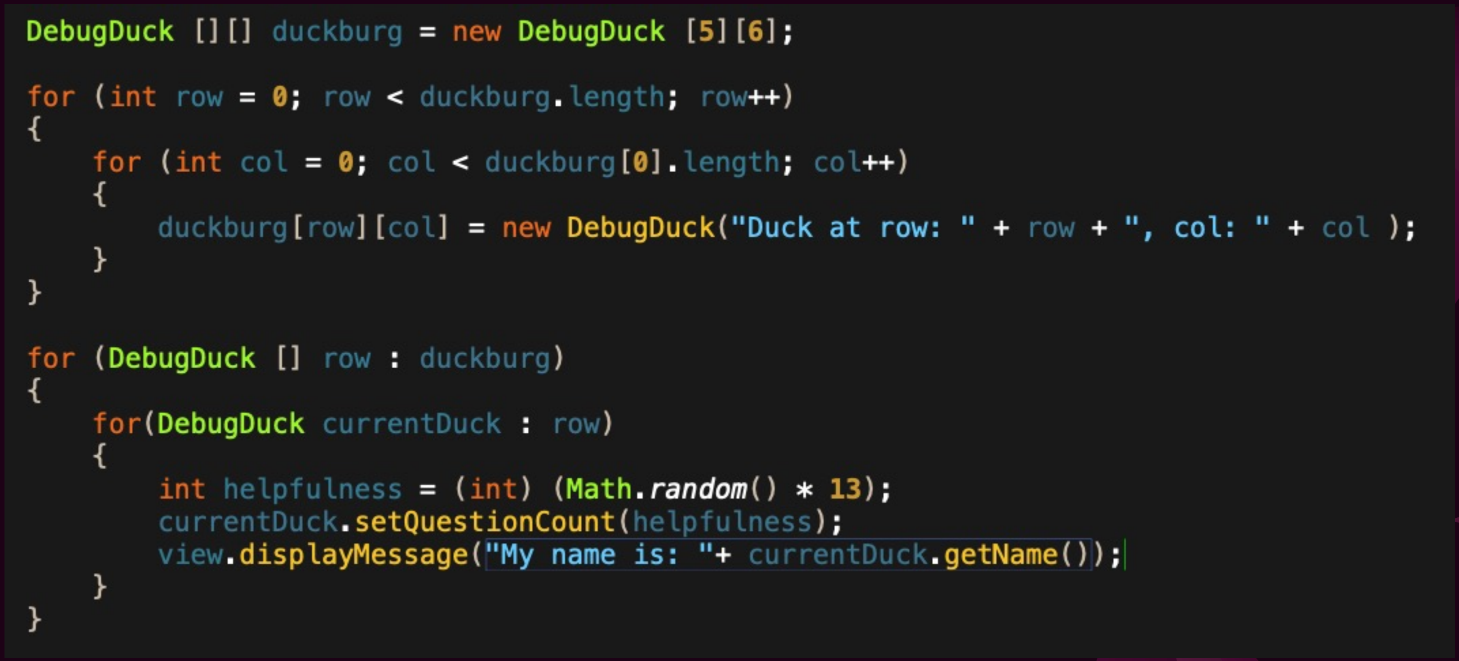


```
for (int [] row : grid)
{
    for (int value : row)
    {
        System.out.println("Current value is: " + value);
    }
}
```



```
for (int [] row : grid)
{
    for (int value : row)
    {
        value = 7;
    }
}
```

Iteration – For Each Demo 2



```
DebugDuck [][] duckburg = new DebugDuck [5][6];

for (int row = 0; row < duckburg.length; row++)
{
    for (int col = 0; col < duckburg[0].length; col++)
    {
        duckburg[row][col] = new DebugDuck("Duck at row: " + row + ", col: " + col );
    }
}

for (DebugDuck [] row : duckburg)
{
    for(DebugDuck currentDuck : row)
    {
        int helpfulness = (int) (Math.random() * 13);
        currentDuck.setQuestionCount(helpfulness);
        view.displayMessage("My name is: "+ currentDuck.getName());
    }
}
```

Bounds Checking

- Be sure to check the correct bounds before doing any logic inside your 2D array
- All row indices are >= 0 and < arrayName.length
- All column indices are >= 0 and < arrayName[0].length
- Do the bounds check BEFORE any other logic or access!!!
  - You want to avoid ArrayIndexOutOfBoundsException

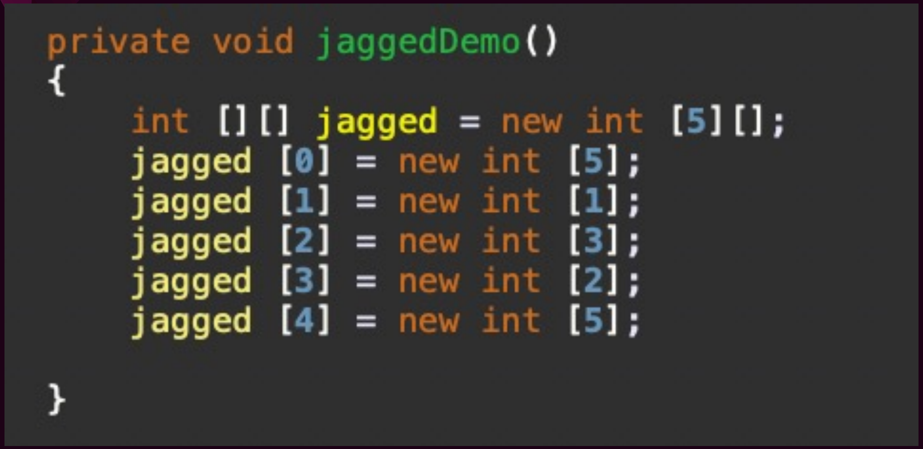
Still the hipster of Java

- No methods belong to the type
- No constructor or () anywhere in sight
- Two public data members
  - arrayName.length for the number of rows
  - arrayName[0].length for the number of columns

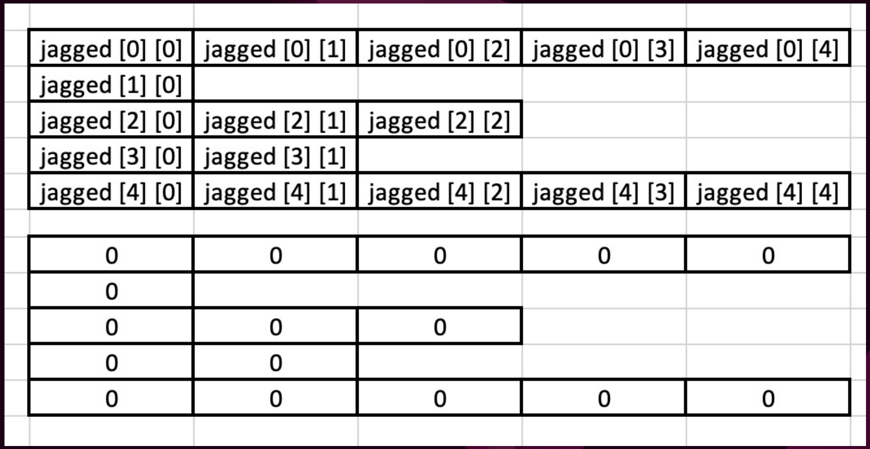
Jagged/Ragged Array

- A non rectangular 2D array
- Each row can have a different amount of columns
- Will NEVER be used in class
- Traversal must NOT use arrayName[0].length since each row’s column count is independent!

Jagged Demo



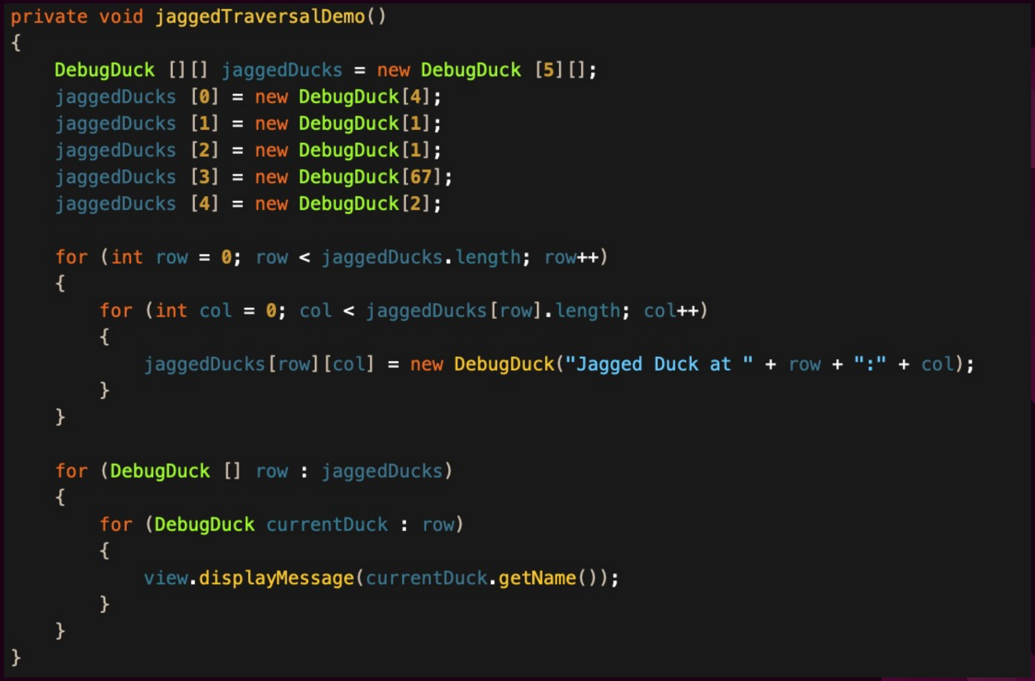
```
private void jaggedDemo()
{
    int [][] jagged = new int [5][];
    jagged [0] = new int [5];
    jagged [1] = new int [1];
    jagged [2] = new int [3];
    jagged [3] = new int [2];
    jagged [4] = new int [5];

}
```



jagged [0] [0]	jagged [0] [1]	jagged [0] [2]	jagged [0] [3]	jagged [0] [4]
jagged [1] [0]				
jagged [2] [0]	jagged [2] [1]	jagged [2] [2]		
jagged [3] [0]	jagged [3] [1]			
jagged [4] [0]	jagged [4] [1]	jagged [4] [2]	jagged [4] [3]	jagged [4] [4]

0	0	0	0	0
0				
0	0	0		
0	0			
0	0	0	0	0

Jagged Traversal



```
private void jaggedTraversalDemo()
{
    DebugDuck [][] jaggedDucks = new DebugDuck [5][];
    jaggedDucks [0] = new DebugDuck[4];
    jaggedDucks [1] = new DebugDuck[1];
    jaggedDucks [2] = new DebugDuck[1];
    jaggedDucks [3] = new DebugDuck[67];
    jaggedDucks [4] = new DebugDuck[2];

    for (int row = 0; row < jaggedDucks.length; row++)
    {
        for (int col = 0; col < jaggedDucks[row].length; col++)
        {
            jaggedDucks[row][col] = new DebugDuck("Jagged Duck at " + row + ":" + col);
        }
    }

    for (DebugDuck [] row : jaggedDucks)
    {
        for (DebugDuck currentDuck : row)
        {
            view.displayMessage(currentDuck.getName());
        }
    }
}
```