2D Arrays

CS 1400 2023
Cody Henrichsen

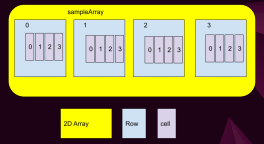
1

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!

2

2D Representation



3

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]

4

2D Element Access

- The row and column are needed for the address of an individual element of a 2D array. The placement of the assignment operator (=) tells you which operation is happening
  - Assignment
    - `arrayName [row][col] = value;`
  - Retrieval
    - `t/Type item = arrayName[row][col];`

5

2D Row Access

- You can access entire rows in a 2D array by the address of the row
- Retrieval
  - `t/Type [] items = arrayName[row];`
- Assignment
  - `arrayName[row] = new t/Type [arrayName.length];`

6

Columns

7

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

8

2D Initialization Demo

```
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"}
                    };
}
```

9

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…

10

Iteration – Standard For Demo

```
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]);
    }
}
```

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

11

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

12

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;
    }
}
```

13

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());
    }
}
```

14

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`

15

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

16

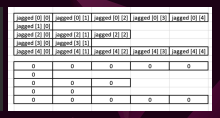
Jagged/Ragged Array

- 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!

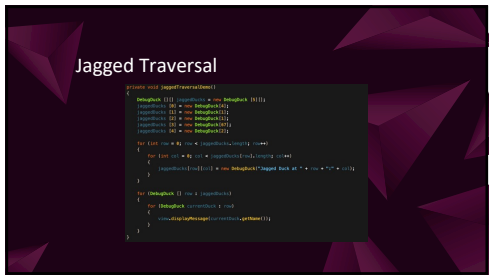
17

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];
}
```



18


Jagged Traversal
private void jaggedTraversalDemo()
{
DebugBuck [][] jaggedBucks = new DebugBuck [5][];
jaggedBucks [0] = new DebugBuck[4];
jaggedBucks [1] = new DebugBuck[1];
jaggedBucks [2] = new DebugBuck[3];
jaggedBucks [3] = new DebugBuck[67];
jaggedBucks [4] = new DebugBuck[2];
for (int row = 0; row < jaggedBucks.Length; row++)
{
for (int col = 0; col < jaggedBucks[row].Length; col++)
{
jaggedBucks[row][col] = new DebugBuck("Jagged Buck at " + row + "," + col);
}
}
for (DebugBuck [] row : jaggedBucks)
{
for (DebugBuck currentBuck : row)
{
view.displayMessage(currentBuck.getName());
}
}
}

19