2D Array

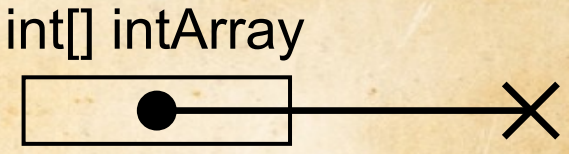
Basics

Lecture Contents

- Review of Array
- 2D Array
  - Declaration
  - Initialization using a literal
  - Accessing elements
    - `for` loop
    - Enhanced `for` loop (for-each loop)
  - Initialization using loops
  - Variable length rows *(not in AP Java Subset)*
- Multidimensional Arrays *(not in AP Java Subset)*

Declaring an Array

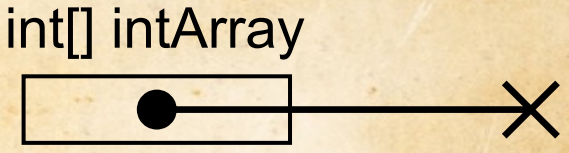
- How do we declare an array, for example, an array of integers. (Just declare the array, do not allocate memory for it.)



Declaring an Array

- Declaring an array only creates the object, it does not allocate memory to store the array.

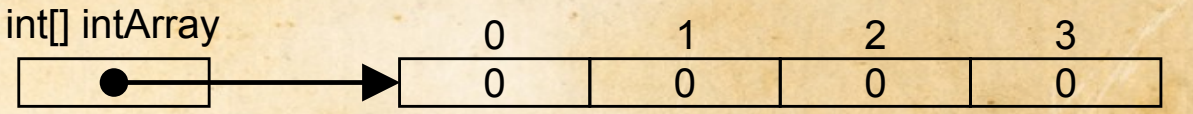
```
int[] intArray;
```



Allocating Memory for an Array

- How can we allocate memory for the array?

```
int[] intArray;
```

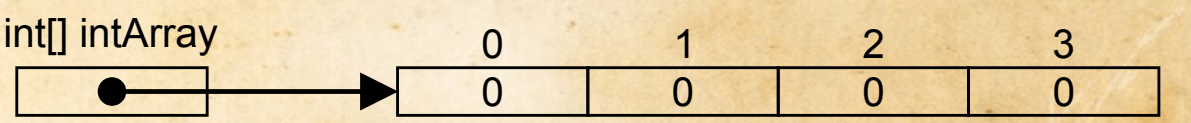


Allocating Memory for an Array

- To allocate memory, we use the keyword `new`.

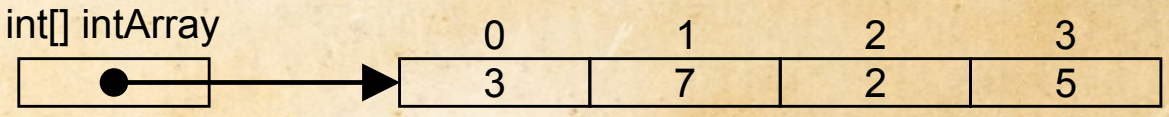
```
int[] intArray;
intArray = new int[4];
```

- `new` will allocate the array and initialize all values to zero.



Initializing an Array

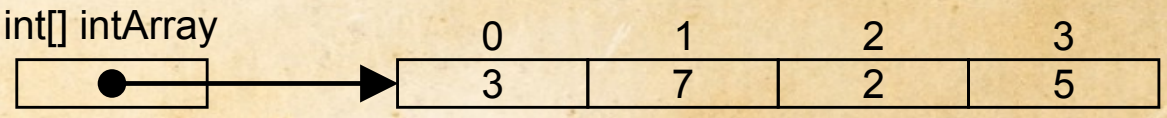
- How can we initialize an array?



Initializing an Array

- We can declare and initialize an array with comma-separated sequence of elements enclosed in curly braces:

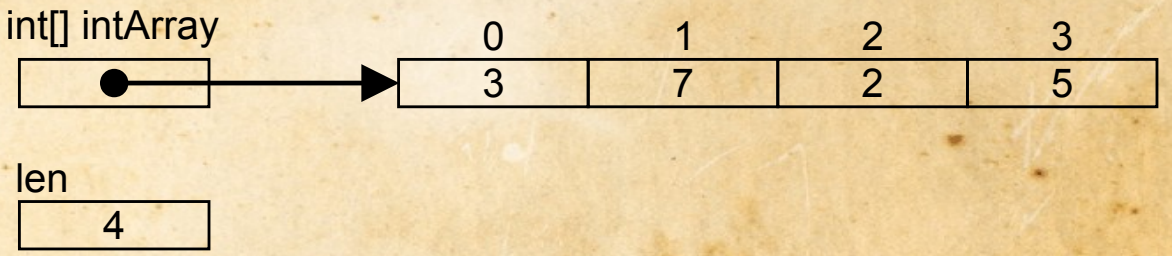
```
int[] intArray = { 3, 7, 2, 5 };
```



Finding the Length of an Array

- How do we determine the number of elements in an array?

```
int[] intArray = { 3, 7, 2, 5 };
```

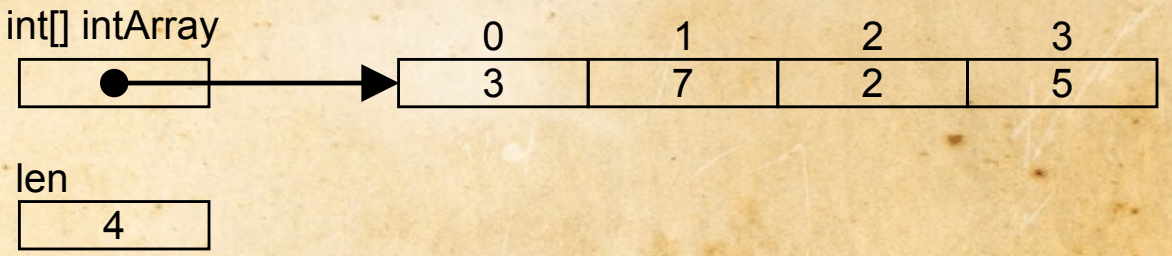


Finding the Length of an Array

- The `length` field gives us the number of elements in an array.

```
int[] intArray = { 3, 7, 2, 5 };
int len = intArray.length;
```

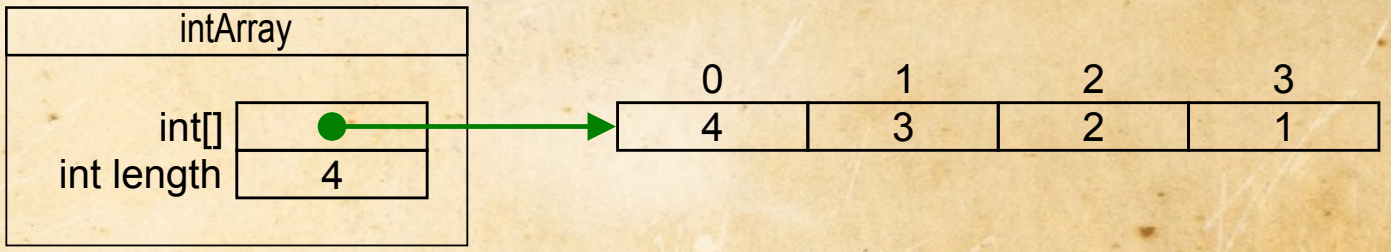
- Notice the largest index is always one less than the value given by the `length` field.



Traversing an Array Sequentially

- We use the `.length` field to traverse an array **forwards**:

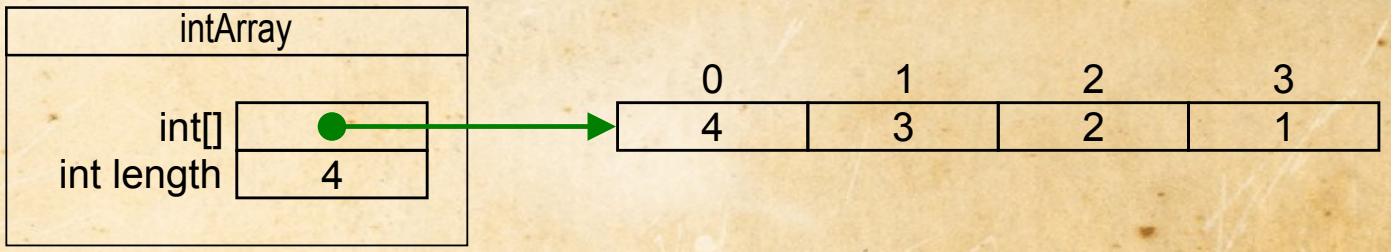
```
int[] intArray = { 3, 7, 2, 5 };
for(int i = 0; i < intArray.length; i++) {
   intArray[i] = 4 - i;
}
```



Traversing an Array Sequentially

- We use the `.length` field to traverse an array **backwards**:

```
int[] intArray = { 3, 7, 2, 5 };
for(int i = intArray.length-1; i>=0 i--) {
   intArray[i] = 4 – i;
}
```



Lecture Contents

- Review of Array ✓
- **2D Array**
  - Declaration and Initialization
  - Initialization using a literal
  - Accessing elements
    - `for` loop
    - Enhanced `for` loop (for-each loop)
  - Initialization using loops
  - Variable length rows *(not in AP Java Subset)*
- Multidimensional Arrays *(not in AP Java Subset)*

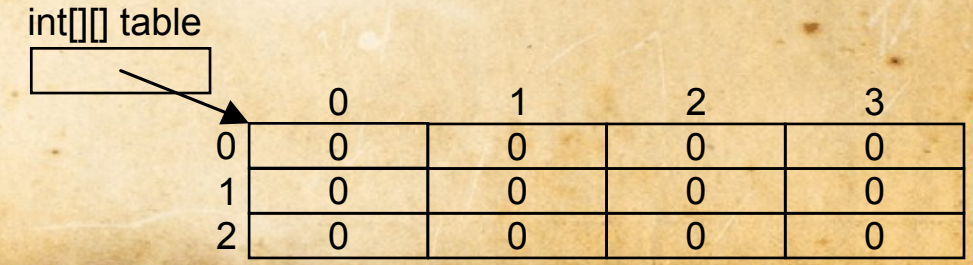
2D Array Declaration and Initialization

- Declaring and initializing an array (one-dimensional)

```
int[] intArray = new int[4];
```

- Declaring and initializing a two-dimensional array, or ***matrix***

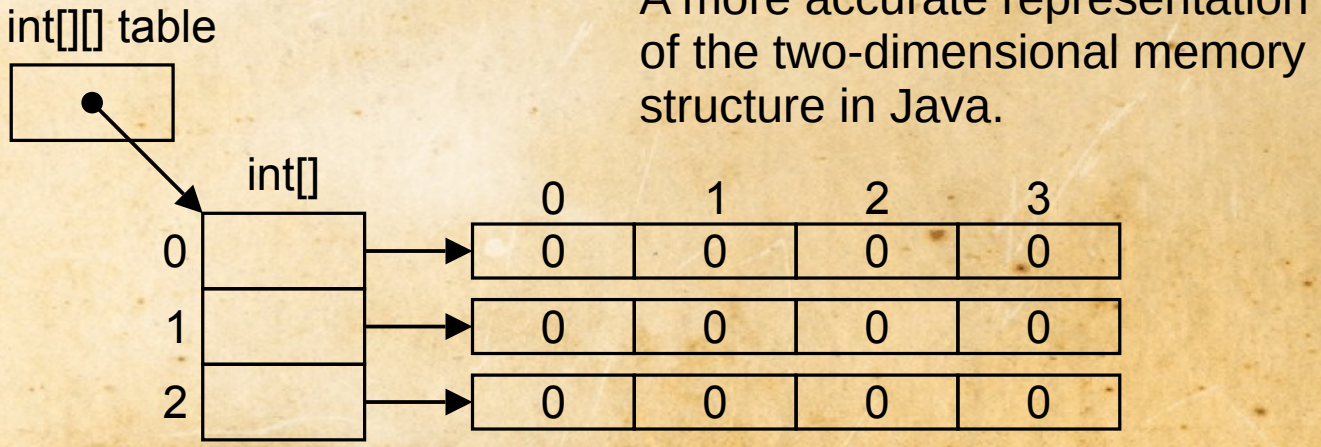
```
final int rows = 3;
final int columns = 4;
int[][] table = new int[rows][columns];
```



2D Array Declaration and Initialization

- Declaring and initializing a two-dimensional array, or ***matrix***

```
final int rows = 3;
final int columns = 4;
int[][] table = new int[rows][columns];
```



A more accurate representation of the two-dimensional memory structure in Java.

2D Array Initialization

- Which are valid:

```
int[][] arr1 = new int[][];
int[][] arr2 = new int[3][];
int[][] arr3 = new int[][4];
int[][] arr4 = new int[3][4];
```

- Let’s go through one-by-one...

2D Array Initialization

- Is this valid?

```
int[][] arr1 = new int[][];
```

2D Array Initialization

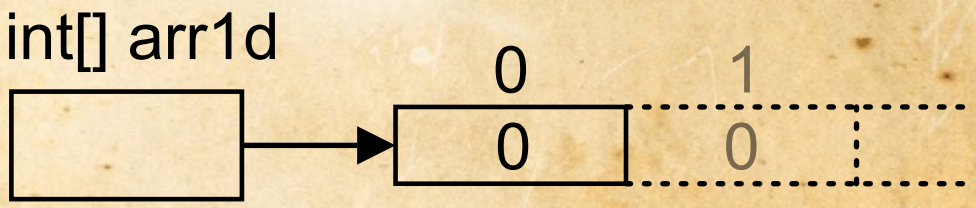
- Is this valid?

✗ `int[][] arr1 = new int[][];`

- No. Just as with a one-dimensional array, this is invalid:

✗ `int[] arr1d = new int[];`

because the compiler has no idea what space needs to be allocated to the array.



2D Array Initialization

- Is this valid?

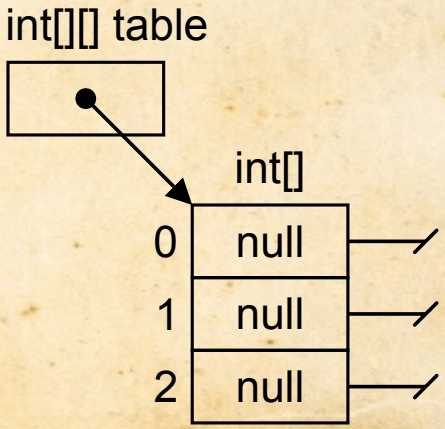
```
int[][] arr2 = new int[3][];
```

2D Array Initialization

- Is this valid?

✓ `int[][] arr2 = new int[3][];`

- Yes. Perhaps surprisingly, but the compiler allocates space for the array of rows. Each `int[]` is set to `null`.



2D Array Initialization

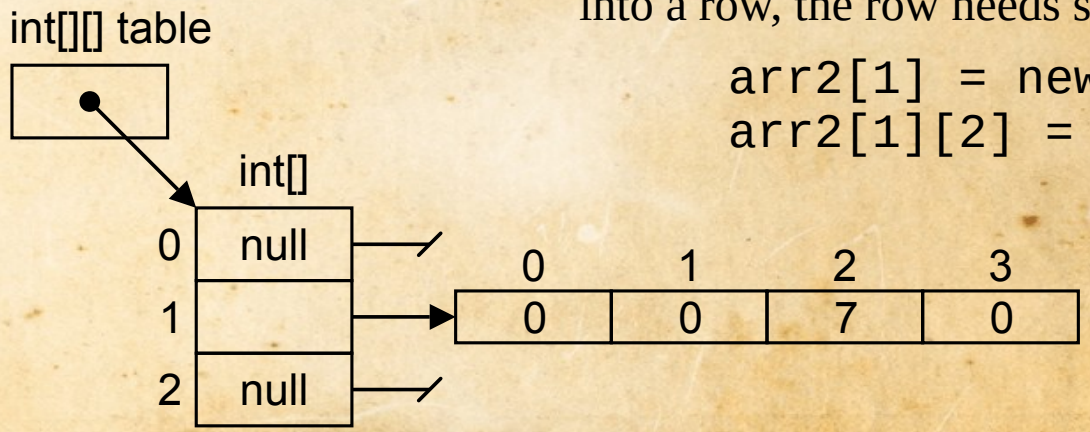
- Is this valid?

✓ `int[][] arr2 = new int[3][];`

- Yes. Perhaps surprisingly, but the compiler allocates space for the array of rows.

Note: before integer values can be inserted into a row, the row needs space allocated:

```
arr2[1] = new int[4];
arr2[1][2] = 7;
```



2D Array Initialization

- Is this valid?

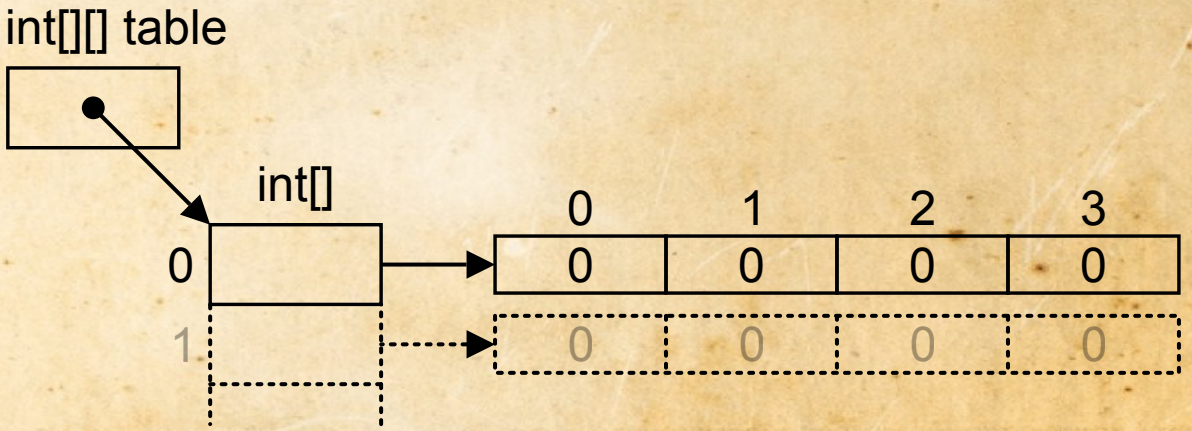
```
int[][] arr3 = new int[][4];
```

2D Array Initialization

- Is this valid?

✗ `int[][] arr3 = new int[][4];`

- No. We are given a row length, but the compiler does not know how many rows to create.



2D Array Initialization

- Is this valid?

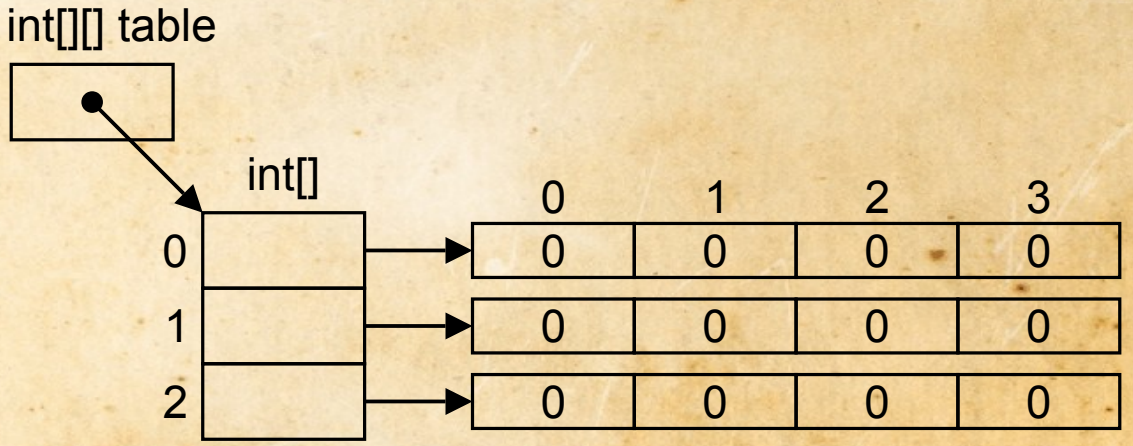
```
int[][] arr4 = new int[3][4];
```

2D Array Initialization

- Is this valid?

✓ `int[][] arr4 = new int[3][4];`

- Yes. The compiler allocates space for the entire two-dimensional array. The `int` in each cell is set to `0` (zero).



2D Array Initialization using a literal

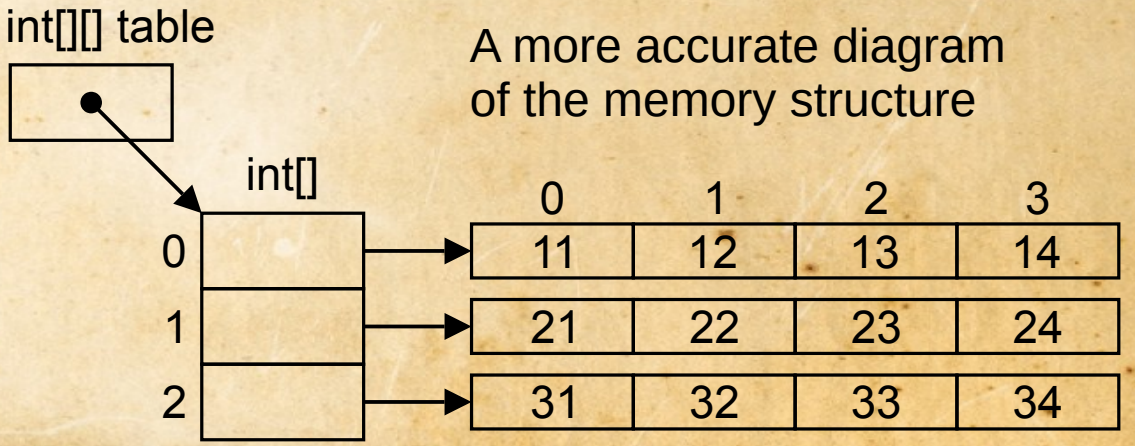
```
int[][] table =
    {
        { 11, 12, 13, 14 }, // row is type int[]
        { 21, 22, 23, 24 },
        { 31, 32, 33, 34 }
    };
```

A conceptual diagram
of the memory structure

	0	1	2	3
0	11	12	13	14
1	21	22	23	24
2	31	32	33	34

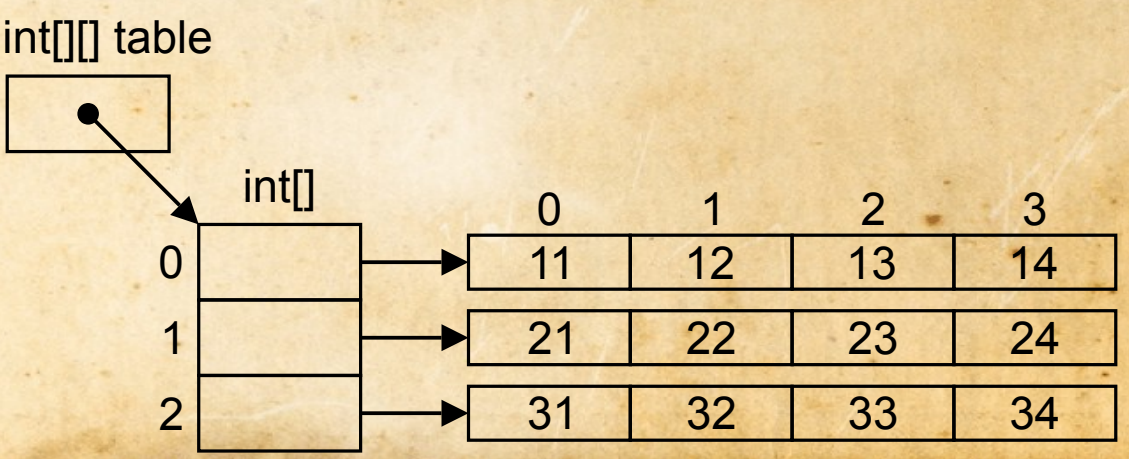
2D Array Initialization using a literal

```
int[][] table =
    {
        { 11, 12, 13, 14 }, // row is type int[]
        { 21, 22, 23, 24 },
        { 31, 32, 33, 34 }
    };
```



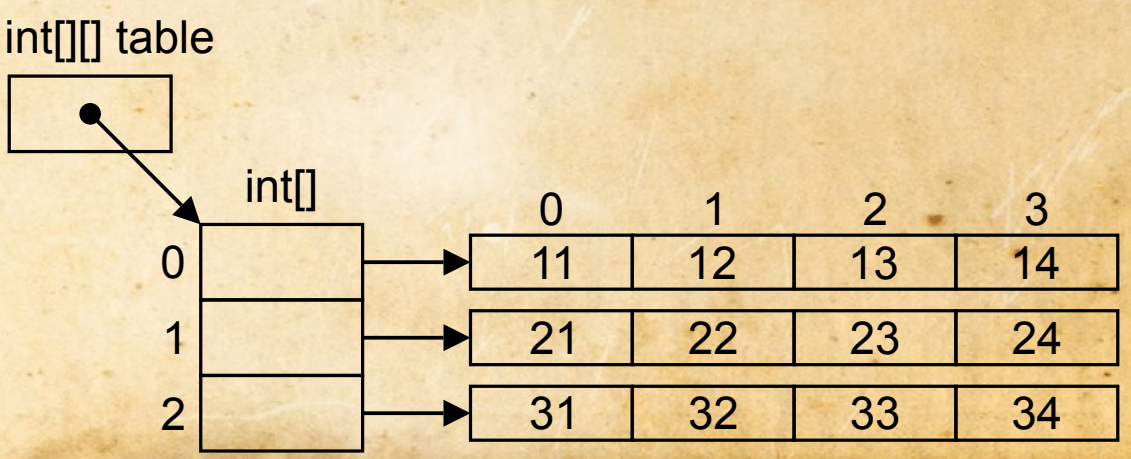
Accessing a 2D Array

```
private static void print2DIntArray(int[][] a) {
    for(int i = 0; i < a.length; i++) {
        for(int j = 0; j < a[i].length; j++) {
            System.out.print(" " + a[i][j]);
        }
        System.out.println();
    }
}
```



Accessing a 2D Array

```
private static void print2DIntArray(int[][] a) {
    for(int[] row: a) {
        for(int column: row ) {
            System.out.print(" " + column);
        }
        System.out.println();
    }
}
```



2D Array Initialization

```
private static void init2DArray(int[][] a) {
    for(int i = 0; i < a.length; i++) {
        for(int j = 0; j < a[i].length; j++) {
            a[i][j] = (i * 10) + j;
        }
    }
}
```

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

2D Array Initialization

```
private static void init2DArray(int[][] a) {
    for(int i = 0; i < a.length; i++) {
        for(int j = 0; j < a[i].length; j++) {
            a[i][j] = (i * 10) + j;
        }
    }
}
```

	0	1	2	3
0	00	01	02	03
1	10	11	12	13
2	20	21	22	23

2D Array Initialization

```
private static void init2DArray(int[][] a) {
    int count = 0;
    for(int[] row : a) {
        for(int column : row) {
            column = count++;
        }
    }
}
```

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

2D Array Initialization

```
private static void init2DArray(int[][] a) {
    int count = 0;
    for(int[] row : a) {
        for(int column : row) {
            column = count++;
        }
    }
}
```

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

Array values unchanged!

2D Array Initialization

```
private static void init2DArray(int[][] a) {
    int count = 0;
    for(int[] row : a) {
        for(int j = 0; j < row.length; j++) {
            row[j] = count++;
        }
    }
}
```

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

2D Array – Variable length rows

Not in AP Java Subset!

```
int[][] table =
    {
        { 11, 12, 13, 14 }, // row is type int[]
        { 21, 22, 23 },     // this row shorter!
        { 31, 32, 33, 34, 35 } // this row longer!
    };
```

	0	1	2	3
0	11	12	13	14
1	21	22	23	
2	31	32	33	34

2D Array – Variable length rows

Not in AP Java Subset!

```
int[][] table =
    {
        { 11, 12, 13, 14 }, // row is type int[]
        { 21, 22, 23 },     // this row shorter!
        { 31, 32, 33, 34, 35 } // this row longer!
    };

   private static void print2DIntArray(int[][] a) {
      for(int i=0; i<a.length; i++) {
         for(int j=0; j<a[i].length; j++) {
            System.out.print(" " + a[i][j]);
         }
         å.out.println();
      }
   }
```

Cannot use `a[0].length` with variable length rows.

Multidimensional Arrays

Not in AP Java Subset!

```
int[][][] a3D =
    {
        {
            { 111, 112, 113, 114 },
            { 121, 122, 123, 124 },
            { 131, 132, 133, 134 }
        },
        {
            { 211, 212, 213, 214 },
            { 221, 222, 223, 224 },
            { 231, 232, 233, 234 }
        }
    };
```

2D Array

Basics