University of Wisconsin
LA CROSSE
Computer Science

Week 05:
Multi-dimensional Arrays

CS 220: Software Design II — D. Mathias

Arrays

Data structure for multiple values

All objects in the array must be of or a subtype of the array's type

Example: Declaring & Instantiating An Array

declare an array of type String called names

```
String[] names;   // both of these lines do the same thing
String names[];
```

instantiate an array of type String with length 6

```
names = new String[6];   // notice we do not use square brackets on left side
```

declare and instantiate an array of type String called names with length 6

```
String[] names = new String[6];
```

Example: Declaring, Instantiating, and Initializing

declare an array of type String called names

```
String[] names;   // both of these lines do the same thing
String names[];
```

instantiate and initialize an array with our name Strings

```
names = {"James", "Amy", "Beth", "Harold", "Remus", "Eliza"};
```

declare, instantiate and initialize an array with our name Strings

```
String[] names = {"James", "Amy", "Beth", "Harold", "Remus", "Eliza"};
```

Example: Array Initialization

initialize an array of type String called names

```
> names[0] = "James";
> names[1] = "Amy";
> names[2] = "Beth";
> names[3] = "Harold";
> names[4] = "Remus";
> names[5] = "Eliza";
>
```

names (String[])

"James" null	"Amy" null	"Beth" null	"Harold" null	"Remus" null	"Eliza" null
0	1	2	3	4	5

Arrays in Memory

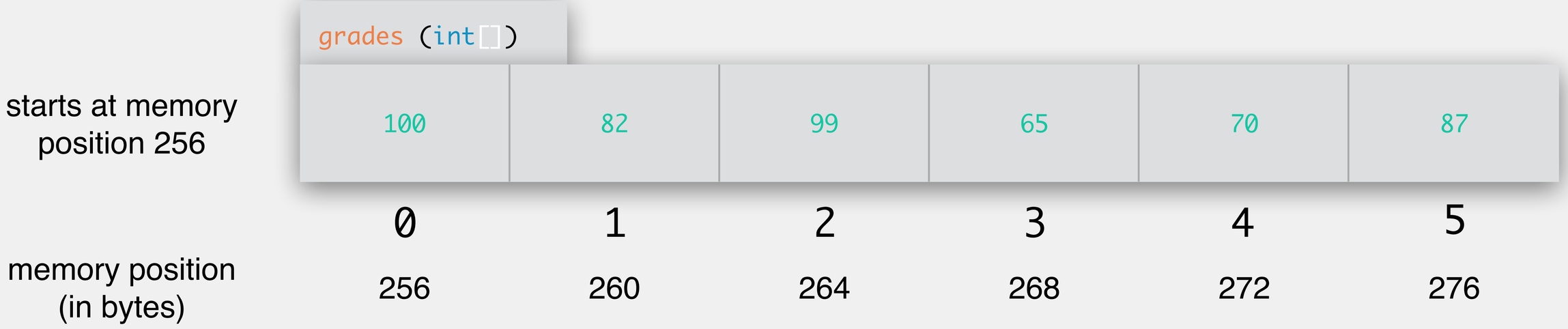
Access is very fast; can skip intermediate positions

Calculated as below:

names[4] = 100;

starting position of the array + (desired index * size of data type)

256 + (4 * 4) = 272



Multidimensional Arrays

- One dimensional arrays can hold data
  - each spot is like a single variable - we hold a single piece of data
- One dimensional arrays can also hold…other arrays
  - must hold the same type of data
  - we'll look at two-dimensional arrays, but can hold as many dimensions as necessary
    - too complex probably indicates a need for a class/object

What Is A 2D Array?

1D array

names (String[])

"James"	"Amy"	"Beth"	"Harold"	"Remus"	"Eliza"
0	1	2	3	4	5

2D array

grades (int[][])

0	92	74	81	83	99	99
1	82	81	90	85	100	99
2	87	80	96	84	89	88
	0	1	2	3	4	5

What Is A 2D Array?

1D array

names (String[])

"James"	"Amy"	"Beth"	"Harold"	"Remus"	"Eliza"
0	1	2	3	4	5

2D array

grades (int[][])

0	92	74	81	83	99	99
1	82	81	90	85	100	99
2	87	80	96	84	89	88
	0	1	2	3	4	5

What Is A 2D Array?

1D array

names (String[])

"James"	"Amy"	"Beth"	"Harold"	"Remus"	"Eliza"
0	1	2	3	4	5

2D array

grades (int[][])

0	92	74	81	83	99	99
1	82	81	90	85	100	99
2	87	80	96	84	89	88
	0	1	2	3	4	5

What Is A 2D Array?

1D array

names (String[])

"James"	"Amy"	"Beth"	"Harold"	"Remus"	"Eliza"
0	1	2	3	4	5

2D array

grades (int[][])

0	92	74	81	83	99	99
1	82	81	90	85	100	99
2	87	80	96	84	89	88
	0	1	2	3	4	5

Example: Declaring & Instantiating An Array

declare an array of type int called grades

```
int[][] grades;   // both of these lines do the same thing
int grades[][];
```

instantiate a 2D array of type int with 3 rows and 6 columns

```
grades = new int[3][6];  // notice we do not use square brackets on left side
```

declare and instantiate an array of type int with 3 rows and 6 columns

```
int[][] grades = new int[3][6];
```

Example: Declaring, Instantiating, and Initializing

declare an array of type int called grades

```
int[][] grades;   // both of these lines do the same thing
int grades[][];
```

instantiate a 2D array of type int with 3 rows and 6 columns

```
grades = new int[3][6];  // notice we do not use square brackets on left side
```

declare, instantiate, and initialize a 2D array of type int with 3 rows and 6 columns

```
int[][] grades = { {92, 74, 81, 83, 99, 99}, {82, 81, 90, 85, 100, 99},
                   {87, 80, 96, 84, 89, 88} };
```

Accessing Values in a 2D Array

```
System.out.println(grades[1][2]);
```

2D array

grades (int[][])

	0	1	2	3	4	5
0	92	74	81	83	99	99
1	82	81	90	85	100	99
2	87	80	96	84	89	88

Accessing Values in a 2D Array

```
System.out.print(grades[1][0] + ", ");
System.out.print(grades[1][1] + ", ");
System.out.print(grades[1][2] + ", ");
System.out.print(grades[1][3] + ", ");
System.out.print(grades[1][4] + ", ");
System.out.print(grades[1][5] + ", ");
```

grades (int[][])

2D array

	0	1	2	3	4	5
0	92	74	81	83	99	99
1	82	81	90	85	100	99
2	87	80	96	84	89	88

Accessing Values in a 2D Array

```
System.out.println(grades[1]);
```

what's wrong with this?

2D array

grades (int[][])

	0	1	2	3	4	5
0	92	74	81	83	99	99
1	82	81	90	85	100	99
2	87	80	96	84	89	88

Accessing Values in a 2D Array

```
System.out.println(grades[1]);
System.out.println(grades);
```

what's wrong with this?

2D array

grades (int[][])

	0	1	2	3	4	5
0	92	74	81	83	99	99
1	82	81	90	85	100	99
2	87	80	96	84	89	88

Accessing Values in a 2D Array

```
for (int i = 0; i < grades[1].length; i++) {
    System.out.print(grades[1][i] + ", ");
}
```

2D array

grades (int[][])

	0	1	2	3	4	5
0	92	74	81	83	99	99
1	82	81	90	85	100	99
2	87	80	96	84	89	88

Exercise: Accessing Values in a 2D Array

```
for (int i = 0; i < grades[1].length; i++) {
    System.out.print(grades[1][i] + ", ");
}
```

2D array

grades (int[][])

	0	1	2	3	4	5
0	92	74	81	83	99	99
1	82	81	90	85	100	99
2	87	80	96	84	89	88

Exercise: Accessing Values in a 2D Array

```
for (int row = 0; row < grades.length; row++) {
    for (int col = 0; col < grades[row].length; col++) {
        System.out.print(grades[row][col] + ", ");
    }
    System.out.println();
}
```

2D array

grades (int[][])

	0	1	2	3	4	5
0	92	74	81	83	99	99
1	82	81	90	85	100	99
2	87	80	96	84	89	88

Ragged Arrays

Might not always want rectangular 2D arrays

Example

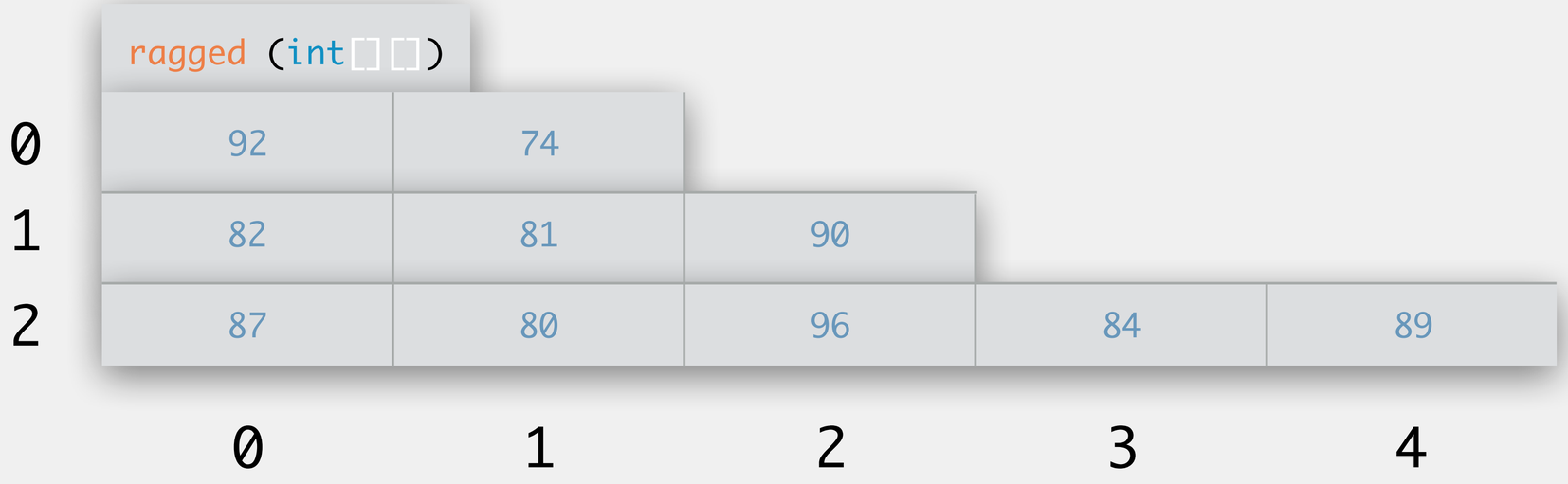
- storing the daily temperature for each month over the course of the year
- should be broken down by month
- `double[][] dailyTemps = new double[12][31];`
- leaves extra space for some months
  - i.e., February, April, June, September, November

Ragged Arrays

With 2D arrays, each spot in the “outer” array points to its own 1D array

- no needs for the 1D arrays to be the same sizes

Need a new way to do initialization



Ragged Arrays: Initialization

```
int[][] ragged = new int[3][];  // notice that the width is left empty
ragged[0] = new int[2];
ragged[1] = new int[3];
ragged[2] = new int[5];
// fills in values
```

ragged (int[][])

	0	1	2	3	4
0	92	74			
1	82	81	90		
2	87	80	96	84	89

Ragged Arrays: Initialization

```
int[][] ragged = new int[3][];  // notice that the width is left empty
ragged[0] = new int[2];
ragged[1] = new int[3];
ragged[2] = new int[5];
// code to fill in values omitted
System.out.println(ragged[1][2]);
System.out.println(ragged[1][3]);
```

ragged (int[][])

	0	1	2	3	4
0	92	74			
1	82	81	90		
2	87	80	96	84	89

Ragged Arrays: Initialization

```
int[][] ragged = new int[3][];  // notice that the width is left empty
ragged[0] = new int[2];
ragged[1] = new int[3];
ragged[2] = new int[5];
// code to fill in values omitted
System.out.println(ragged[1][2]);  // will print 90
System.out.println(ragged[1][3]);  // throws ArrayIndexOutOfBoundsException
```

ragged (int[][])

	0	1	2	3	4
0	92	74			
1	82	81	90		
2	87	80	96	84	89

Exercise: Ragged Arrays

Given the array below, write code to print out every value, with each row on its own line.

```
ragged (int[][])
```

	0	1	2	3	4
0	92	74			
1	82	81	90		
2	87	80	96	84	89

Exercise: Ragged Arrays

```
> for (int row = 0; row < ragged.length; row++) {
      for (int col = 0; col < ragged[row].length; col++) {
          System.out.print(ragged[row][col] + " ");
      }
      System.out.println();
  }
```

ragged (int[][])

	0	1	2	3	4
0	92	74			
1	82	81	90		
2	87	80	96	84	89

Exercise: Ragged Arrays

```
for (int row = 0; row < ragged.length; row++) {
    for (int col = 0; col < ragged[row].length; col++) {
        System.out.print(ragged[row][col] + " ");
    }
    System.out.println();
}
```

row (int)
0

ragged (int[][])

	0	1	2	3	4
0	92	74			
1	82	81	90		
2	87	80	96	84	89

Exercise: Ragged Arrays

```
for (int row = 0; row < ragged.length; row++) {
    for (int col = 0; col < ragged[row].length; col++) {
      > System.out.print(ragged[row][col] + " ");
    }
    System.out.println();
}
```

row (int)
0

col (int)
0

ragged (int[][])

	0	1	2	3	4
0	92	74			
1	82	81	90		
2	87	80	96	84	89

Exercise: Ragged Arrays

```
for (int row = 0; row < ragged.length; row++) {
    for (int col = 0; col < ragged[row].length; col++) {
        System.out.print(ragged[row][col] + " ");
    }
    System.out.println();
}
```

row (int)
0

col (int)
0

```
92
```

ragged (int[][])

	0	1	2	3	4
0	92	74			
1	82	81	90		
2	87	80	96	84	89

Exercise: Ragged Arrays

```
for (int row = 0; row < ragged.length; row++) {
    for (int col = 0; col < ragged[row].length; col++) {
      > System.out.print(ragged[row][col] + " ");
    }
    System.out.println();
}
```

row (int)
0

col (int)
1

```
92
```

ragged (int[][])

	0	1	2	3	4
0	92	74			
1	82	81	90		
2	87	80	96	84	89

Exercise: Ragged Arrays

```
for (int row = 0; row < ragged.length; row++) {
    for (int col = 0; col < ragged[row].length; col++) {
        System.out.print(ragged[row][col] + " ");
    }
    System.out.println();
}
```

row (int)
0

col (int)
2

```
92 74
```

ragged (int[][])

	0	1	2	3	4
0	92	74			
1	82	81	90		
2	87	80	96	84	89

Exercise: Ragged Arrays

```
for (int row = 0; row < ragged.length; row++) {
    for (int col = 0; col < ragged[row].length; col++) {
        System.out.print(ragged[row][col] + " ");
    }
> System.out.println();
}
```

row (int)
0

```
92 74
```

ragged (int[][])

	0	1	2	3	4
0	92	74			
1	82	81	90		
2	87	80	96	84	89

Exercise: Ragged Arrays

```
> for (int row = 0; row < ragged.length; row++) {
      for (int col = 0; col < ragged[row].length; col++) {
          System.out.print(ragged[row][col] + " ");
      }
      System.out.println();
  }
```

row (int)
0

```
92 74
```

ragged (int[][])

	0	1	2	3	4
0	92	74			
1	82	81	90		
2	87	80	96	84	89

Exercise: Ragged Arrays

Write the code to declare and instantiate a 2D array to hold a temperature (`double`) for every day of every month. Each month should be its own row in the array. (i.e., row 0 is for January, row 1 for February, etc.)

Once you believe you have a correct declaration, write loops to ask the user, via the console, for a temperature for every day.

split

arguments: a single String pattern to split on (we'll usually do a single char)

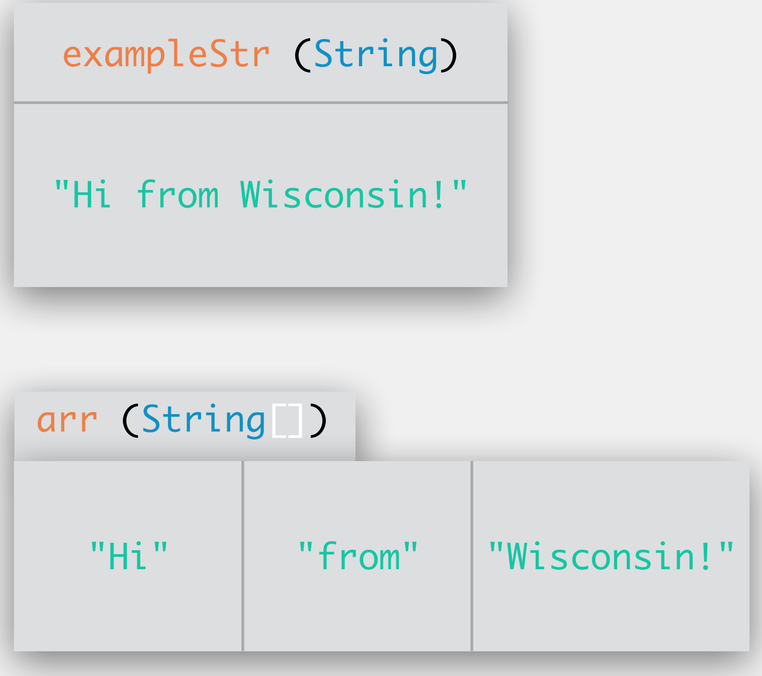
returns: a String array containing each split component, in order

```
<String>.split(<String>);
```

```
>String exampleStr = "Hi from Wisconsin!";

>String[] arr = exampleStr.split(" ");
>
```

memory



Exercise: Ragged Arrays

Write the code to read in sentences from a text file and store them word by word in a 2D array. Each sentence is on its own line, and each row should be one sentence. The number of sentences in the file is listed at the top of the file.

```
4
An example sentence
A few more examples below
Using 2D arrays can be tricky
Fin
```

toCharArray

arguments: nothing

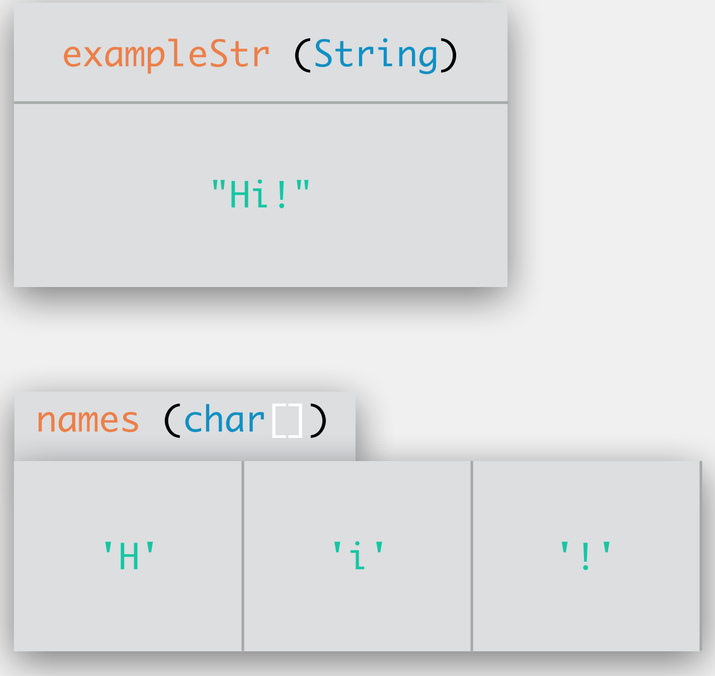
returns: a char array containing each character in the String, in order

```
<String>.toCharArray();
```

```
>String exampleStr = "Hi!";

>char[] arr = exampleStr.toCharArray();
>
```

memory



Exercise: Ragged Arrays

Write the code to read in sentences from a file and store them character by character in a 2D array. Each sentence is on its own line, and each row should be one sentence. The number of sentences in the file is at the top of the file.

```
4
An example sentence
A few more examples below
Using 2D arrays can be tricky
Fin
```