Module 3: Arrays

Ron K. Cytron

Department of Computer Science and Engineering
Washington University in Saint Louis
Thanks to Alan Waldman
for comments that improved these slides

Prepared for 2u
Semester Online

Copyright Ron K. Cytron 2013
3.0 Introduction

- Consider now a larger problem than we've seen before
  - Compute the average of 10 temperature readings
- We could declare 10 different variables and express the computation that way
  - \((\text{temp0} + \text{temp1} + \ldots + \text{temp9})/10\)
- But as we scale to 100, 1000 readings, typing in so many variables becomes tedious
- Arrays allow us a uniform and relatively terse treatment of many related variables
  - For example, 1000 temperature readings
  - These are necessarily of the same type \(\text{double}\)
- We instead instruct the computer to provide for 1000 temperature readings, each of type \(\text{double}\)
- We can then iterate over the array and compute the average of the array's entries
- Formally, an array is a mapping from
  - An \(\text{int}\) (which reading do I want?), which is the index into the array
  - To a \(\text{double}\) (the actual reading), which is the value stored at the indexed location
- A big "gotcha" here is that arrays start with the index 0, not 1
3.1 Examples

double
    t0 = 50.3,
    t1 = 34.4,
    t2 = 39.0,
    t3 = 46.7,
    t4 = 50.9;

double avg =
    (t0 + t1 + t2 + t3 + t4) / 5;

System.out.println("Average is " + avg);
3.1 Examples

double

\begin{align*}
  t0 &= 50.3, \\
  t1 &= 34.4, \\
  t2 &= 39.0, \\
  t3 &= 46.7, \\
  t4 &= 50.9;
\end{align*}

Tedious! Let's try iteration to compute the average.

\begin{verbatim}
\begin{verbatim}
    double avg =
    (t0 + t1 + t2 + t3 + t4) / 5;
\end{verbatim}
\end{verbatim}

\texttt{System.out.println("Average is ") + avg);}


double
    t0 = 50.3,
    t1 = 34.4,
    t2 = 39.0,
    t3 = 46.7,
    t4 = 50.9;

double sum = 0;
for (int i=0; i < 5; ++i) {

}
double ans = sum / 5;
System.out.println("The average is " + ans);
3.1 Examples

double
t0 = 50.3,
t1 = 34.4,
t2 = 39.0,
t3 = 46.7,
t4 = 50.9;

double sum = 0;
for (int i=0; i < 5; ++i) {
    sum = sum + ___________;
}
double ans = sum / 5;
System.out.println("The average is " + ans);

We have no way to name the i\textsuperscript{th} element (t0, t1, etc.)
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

    The variable t is declared as an array of double values. The values are specified directly between the braces. Thus there are 5 elements in the array.

double sum = 0;
for (int i=0; i < 5; ++i) {
}

double ans = sum / 5;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

double sum = 0;
for (int i=0; i < 5; ++i) {
}

double ans = sum / 5;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,  \textbf{t[1]}  
    39.0,
    46.7,
    50.9};

double sum = 0;
for (int i=0; i < 5; ++i) {
    sum += t[i];
}

double ans = sum / 5;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0, \(t[2]\)
    46.7,
    50.9};

double sum = 0;
for (int i=0; i < 5; ++i) {
}
double ans = sum / 5;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,  \textcolor{cyan}{t[3]}
    50.9};

double sum = 0;
for (int i=0; i < 5; ++i) {
}

double ans = sum / 5;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
  50.3,
  34.4,
  39.0,
  46.7,
  50.9};

double sum = 0;
for (int i=0; i < 5; ++i) {
}

double ans = sum / 5;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

double sum = 0;
for (int i=0; i < 5; ++i) {
    sum = sum + t[i];
}
double ans = sum / 5;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

It would be nice if we did not have to remember the size of the array ourselves.

It would be easy to miscount and make a mistake in this computation.

double sum = 0;
for (int i=0; i < 5; ++i) {
    sum = sum + t[i];
}

double ans = sum / 5;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

It would be nice if we did not have to remember the size of the array ourselves.

The number of entries in an array t can be found by specifying t.length as shown

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / 5;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

It would be nice if we did not have to remember the size of the array ourselves.
The number of entries in an array t can be found by specifying t.length as shown

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

  sum     i     t[i]
  -------  ----  -----
     84.7    1    34.4

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is "+ ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

    sum  |  i  |  t[i]  
   ---  |----|-------
  123.7 | 3  | 46.7  

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}

double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

<table>
<thead>
<tr>
<th>sum</th>
<th>i</th>
<th>t[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>170.4</td>
<td>3</td>
<td>46.7</td>
</tr>
</tbody>
</table>

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}

double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}

double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
  50.3,
  34.4,
  39.0,
  46.7,
  50.9};

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}

double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = {
    50.3,
    34.4,
    39.0,
    46.7,
    50.9};

    | sum | ans |
    |----|----|
    | 221.3 | 44.26 |

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t =
   
   // We want t to be an array of 1000 values,  
   // chosen at random
   // We can still compute the average  
   // as shown below

double sum = 0;
for (int i=0; i < t.length; ++i) {
   sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = new double[1000];
//
// This generates an array of 1000 values, each initially 0
//
//
double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = new double[1000];

This generates an array of 1000 values, each initially 0

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = new double[1000];
//
for (int i=0; i < t.length; ++i) {
    // This iterates over each index of the array
}
//

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);

<table>
<thead>
<tr>
<th>k</th>
<th>t[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>0</td>
</tr>
<tr>
<td>999</td>
<td>0</td>
</tr>
</tbody>
</table>
3.1 Examples

double[] t = new double[1000];
/
for (int i=0; i < t.length; ++i) {
	t[i] = Math.random();
}  Assigns each a random value
/

double sum = 0;
for (int i=0; i < t.length; ++i) {
	sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = new double[1000];  //
for (int i=0; i < t.length; ++i) {  
  t[i] = Math.random();
}

//

double sum = 0;
for (int i=0; i < t.length; ++i) {  
  sum = sum + t[i];
}

double ans = sum / t.length;
System.out.println("The average is " + ans);
double[] t = new double[1000];
//
for (int i=0; i < t.length; ++i) {
    t[i] = Math.random();
}
//

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = new double[1000];

for (int i=0; i < t.length; ++i) {
    t[i] = Math.random();
}

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}

double ans = sum / t.length;
System.out.println("The average is "+ ans);
3.1 Examples

double[] t = new double[1000];
//
for (int i=0; i < t.length; ++i) {
    t[i] = Math.random();
}
//

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = new double[1000];
//
for (int i=0; i < t.length; ++i) {
    t[i] = Math.random();
}
//

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = new double[1000];

for (int i=0; i < t.length; ++i) {
    t[i] = Math.random();
}

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = new double[1000];

for (int i=0; i < t.length; ++i) {
    t[i] = Math.random();
}

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}

double ans = sum / t.length;
System.out.println("The average is " + ans);
3.1 Examples

double[] t = new double[1000];
//
for (int i=0; i < t.length; ++i) {
    t[i] = Math.random();
}
//

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is " + ans);

Many iterations later.....
3.1 Examples

double[] t = new double[1000];
    //
    for (int i=0; i < t.length; ++i) {
        t[i] = Math.random();
    }
    //
    double sum = 0;
    for (int i=0; i < t.length; ++i) {
        sum = sum + t[i];
    }
    double ans = sum / t.length;
    System.out.println("The average is " + ans);

<table>
<thead>
<tr>
<th>k</th>
<th>t[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.324</td>
</tr>
<tr>
<td>1</td>
<td>0.989</td>
</tr>
<tr>
<td>2</td>
<td>0.131</td>
</tr>
<tr>
<td>3</td>
<td>0.494</td>
</tr>
<tr>
<td>4</td>
<td>0.434</td>
</tr>
<tr>
<td>5</td>
<td>0.501</td>
</tr>
<tr>
<td>6</td>
<td>0.547</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>999</td>
<td>0</td>
</tr>
</tbody>
</table>
3.1 Examples

double[] t = new double[1000];

for (int i=0; i < t.length; ++i) {
    t[i] = Math.random();
}

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}

double ans = sum / t.length;
System.out.println("The average is "+ ans);
3.1 Examples

double[] t = new double[1000];
//
for (int i=0; i < t.length; ++i) {
    t[i] = Math.random();
}
//

double sum = 0;
for (int i=0; i < t.length; ++i) {
    sum = sum + t[i];
}
double ans = sum / t.length;
System.out.println("The average is "+ ans);
3.2 Syntax and usage

- Declaring an array

```java
String[] names = { "Bob","Carol","Alice" };
```
3.2 Syntax and usage

• Declaring an array

```java
String[] names = { "Bob","Carol","Alice" };
```

– The type of each element contained in the array, followed by an opening and closing bracket

– Either

  • List the array elements in braces
  • Use new to provision for elements you will assign later
3.2 Syntax and usage

• Declaring an array

```java
String[] names = { "Bob","Carol","Alice" };
```

– The type of each element contained in the array, followed by an opening and closing bracket

– Either

  • List the array elements in braces
  • Use new to provision for elements you will assign later
3.2 Syntax and usage

• Declaring an array

```java
String[] names = new String[131];
```

– The type of each element contained in the array, followed by an opening and closing bracket

– Either

  • List the array elements in braces
  • Use `new` to provision for elements you will assign later.
3.2 Syntax and usage

• Declaring an array
  
  ```java
  String[] names = new String[131];
  ```

  – The type of each element contained in the array, followed by an opening and closing bracket

  – Either

    • List the array elements in braces
    • Use `new` to provision for elements you will assign later.
      
      – The elements have a default initial value
        
        » 0 for ints, 0.0 for doubles, null for Strings
3.2 Syntax and usage

• Referencing elements of an array
  
  names[ i ]

  – References the i\textsuperscript{th} element of the names array
  – In place of i, any integer-valued expression can be used
  – The elements of the array start with index 0
    • And therefore stop just 1 short of the length of the array
  – String[ ] names = { "Alice", "Bob", "Carol" };
    • names[0] has "Alice"
    • names[1] has "Bob"
    • names[2] has "Carol"
    • names.length is 3
3.2 Syntax and usage

• Consider the code:

```java
String[] names = {"Alice","Bob","Carol"};
```

<table>
<thead>
<tr>
<th>Index i</th>
<th>names[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;Alice&quot;</td>
</tr>
<tr>
<td>1</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Carol&quot;</td>
</tr>
</tbody>
</table>
3.2 Syntax and usage

• Consider the code:

```java
String[] names = {"Alice","Bob","Carol"};
names[2] = names[0];
```
3.2 Syntax and usage

- Consider the code:

```java
String[] names = {"Alice","Bob","Carol"};
names[2] = names[0];
```

<table>
<thead>
<tr>
<th>Index i</th>
<th>names[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;Alice&quot;</td>
</tr>
<tr>
<td>1</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Carol&quot;</td>
</tr>
</tbody>
</table>

The right-hand side is evaluated
3.2 Syntax and usage

• Consider the code:

```java
String[] names = {"Alice","Bob","Carol"};
names[2] = names[0];
```

The result is stored at the location specified on the left-hand side.
3.2 Syntax and usage

• Consider the code:

```java
String[] names = {"Alice","Bob","Carol"};
names[2] = names[0];
names[0] = names[1];
```
### 3.2 Syntax and usage

- **Consider the code:**

```java
String[] names = {"Alice","Bob","Carol"};
names[2] = names[0];
names[0] = names[1];
names[2] = names[0] + names[1];
```

<table>
<thead>
<tr>
<th>Index i</th>
<th>names[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>1</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Alice&quot;</td>
</tr>
</tbody>
</table>

The right-hand side is evaluated.
The result is "Bob"+"Bob"
or "BobBob"
3.2 Syntax and usage

• Consider the code:

```java
String[] names = {"Alice","Bob","Carol"};
names[2] = names[0];
names[0] = names[1];
names[2] = names[0] + names[1];
```

<table>
<thead>
<tr>
<th>Index i</th>
<th>names[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>1</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;BobBob&quot;</td>
</tr>
</tbody>
</table>

The result is then stored at the location specified on the left-hand side.
3.2 Syntax and usage

• Consider the code:

```java
String[] names = {"Alice","Bob","Carol"};
names[2] = names[0];
names[0] = names[1];
names[2] = names[0] + names[1];
names[0] = names[1] + " Loblah";
```

The right-hand side is evaluated
The result is
"Bob"+" Loblah"
or
"Bob Loblah"

<table>
<thead>
<tr>
<th>Index i</th>
<th>names[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>1</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;BobBob&quot;</td>
</tr>
</tbody>
</table>
3.2 Syntax and usage

- Consider the code:

```java
String[] names = {"Alice","Bob","Carol");
names[2] = names[0];
names[0] = names[1];
names[2] = names[0] + names[1];
names[0] = names[1] + " Loblah";
```

<table>
<thead>
<tr>
<th>Index i</th>
<th>names[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;Bob Loblah&quot;</td>
</tr>
<tr>
<td>1</td>
<td>&quot;Bob&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;BobBob&quot;</td>
</tr>
</tbody>
</table>

End of Oyster
3.3 Exercise

• **Video intro**
  – *It is often useful to make a copy of an array*
    • Keep the original data
    • Change the copy as we like

• **Example**
  – A names array containing "Alice", "Bob", "Carol"
  – A copy of that array where each entry is a letter's greeting, as in "Dear Alice,"
  – Another copy might have the close and signature, as in "Best regards, Alice"
3.3 Exercise

• **Question card**
  – **Declare a simple array called `names` with 5 strings**
    • Use any strings you like
    • Which syntax for declaring arrays is preferable for the `names` array?
  – **Now write code that declares another array called `copy` and arrange for that array to be a copy of the `names` array**
    • Even though our arrays are small, pretend at this point that the `names` array is really large
    • Which syntax for declaring arrays is preferable for the `copy` array?
  – **Try out your code and make sure that it works**
3.3 Exercise

• Question card
  – Now change your code so that the copy contains the *pig latin* of the original array.
    • Do this by appending the letters "ay" at the end of each entry.
    • Do not worry about whether the entry begins with a vowel or not, just append "ay"
  – Try out your code and make sure that it works
3.3 Exercise

• **Video response**
  – Go over the code for the solution
  – Highlight the different ways arrays are initialized
  – Ask and answer what would happen if the array instantiation were inside the loop instead of outside?
3.4 Permuting an array

- Suppose an array represents a deck of cards, as shipped from the manufacturer
  - In canonical order
    - 2 clubs
    - 3 clubs
    - ...
    - Ace clubs
    - 2 diamonds
    - ...
    - Ace spades

- To play a game we need to permute the array (shuffle the deck). How can we do this?
3.4 Permuting an array

• Although it is difficult for humans, a computer can shuffle a deck *perfectly*

_________  A
_________  B
_________  C
_________  D
_________  E
_________  F
_________  G
_________  H
3.4 Permuting an array

- Although it is difficult for humans, a computer can shuffle a deck *perfectly*.
3.4 Permuting an array

- Although it is difficult for humans, a computer can shuffle a deck *perfectly*
3.4 Permuting an array

• Although it is difficult for humans, a computer can shuffle a deck *perfectly*
3.4 Permuting an array

• Although it is difficult for humans, a computer can shuffle a deck perfectly

After 1 perfect shuffle
Notice the top and bottom card do not move!

Let's do another round…
3.4 Permuting an array

• Although it is difficult for humans, a computer can shuffle a deck *perfectly*

_________  A
_________  E
_________  B
_________  F

----------  C
----------  G
----------  D
----------  H
3.4 Permuting an array

• Although it is difficult for humans, a computer can shuffle a deck *perfectly*
3.4 Permuting an array

• Although it is difficult for humans, a computer can shuffle a deck *perfectly*

__________  A
__________  C
__________  E
__________  G
__________  B
__________  D
__________  F
__________  H
3.4 Permuting an array

• Although it is difficult for humans, a computer can shuffle a deck *perfectly*

<table>
<thead>
<tr>
<th></th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>G</td>
</tr>
<tr>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>F</td>
</tr>
<tr>
<td></td>
<td>H</td>
</tr>
</tbody>
</table>

After 2 perfect shuffles

Let's do another round…

*(can you see what will happen?)*
3.4 Permuting an array

- Although it is difficult for humans, a computer can shuffle a deck *perfectly*
3.4 Permuting an array

- Although it is difficult for humans, a computer can shuffle a deck *perfectly*

A C E G
B D F H
3.4 Permuting an array

- Although it is difficult for humans, a computer can shuffle a deck *perfectly*

```
        A
       ___
      ___  ___
     ___  ___  ___
    ___  ___  ___  ___
   ___  ___  ___  ___  ___
```

```
        B
       ___
      ___  ___
     ___  ___  ___
    ___  ___  ___  ___
   ___  ___  ___  ___  ___
```

```
        C
       ___
      ___  ___
     ___  ___  ___
    ___  ___  ___  ___
   ___  ___  ___  ___  ___
```

```
        D
       ___
      ___  ___
     ___  ___  ___
    ___  ___  ___  ___
   ___  ___  ___  ___  ___
```

```
        E
       ___
      ___  ___
     ___  ___  ___
    ___  ___  ___  ___
   ___  ___  ___  ___  ___
```

```
        F
       ___
      ___  ___
     ___  ___  ___
    ___  ___  ___  ___
   ___  ___  ___  ___  ___
```

```
        G
       ___
      ___  ___
     ___  ___  ___
    ___  ___  ___  ___
   ___  ___  ___  ___  ___
```

```
        H
       ___
      ___  ___
     ___  ___  ___
    ___  ___  ___  ___
   ___  ___  ___  ___  ___
```
3.4 Permuting an array

- Although it is difficult for humans, a computer can shuffle a deck *perfectly*

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>We are back where we started!</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After 3 perfect shuffles

Puzzler: how many rounds would it take for 16 cards to return to their original order?

Also: How would you formalize the notion of reasonable shuffling, and how does that formalization reveal the flaw in this shuffling algorithm?
• How should we think about how many times to shuffle 16 cards?
  – We can try it with 16, seems hard
• Or we could develop intuition by trying smaller decks
  – 4
  – 2
• Turn to camera, what do you think?
• What do we at roundtable see from this?
  – 2 1
  – 4 2
  – 8 3
  – 16 ??
• Large number of cards (say 64) only 6 possible decks!
Roundtable 4c

• What does it mean for the cards in the deck to be randomized?
  – What do you think?
  – What does live student think?

• We have seen Math.random()
  – How does it work?
  – What makes it random?

• Las Vegas
  – what would happen if the machines were not random?

• Definition of pseudorandom
3.4 Permuting an array

- Although it is difficult for humans, a computer can shuffle a deck *perfectly*

Perfect shuffling is *too perfect*

We cannot shuffle mechanically by imitating what humans do

We could not fly until we stopped imitating bird flight
3.4 Permuting an array

• Let's consider a different algorithm, based on choosing cards randomly

_________  A
_________  B
_________  C
_________  D
_________  E
_________  F
_________  G
_________  H

Pick a card, any card!
3.4 Permuting an array

- Let's consider a different algorithm, based on choosing cards randomly.

---

Pick a card, any card!

A
B
C
D
E
F
G
H
3.4 Permuting an array

- Let's consider a different algorithm, based on choosing cards randomly

```
[ ] A  
[ ] B  Pick a card, any card!
[ ] D  Put it on the table, so the first card chosen is the bottommost card
[ ] E
[ ] F
[ ] G
[ ] H  
[ ] C
```
3.4 Permuting an array

• Let's consider a different algorithm, based on choosing cards randomly

A
B
D
E
F
G
H

Pick a card, any card!
Put it on the table, so the first card chosen is the bottommost card
Then close up the hole in the deck
3.4 Permuting an array

- Let's consider a different algorithm, based on choosing cards randomly.

Pick a card, any card!

Put it on the table, so the first card chosen is the bottommost card.

Then close up the hole in the deck.

A
B
D
E
F
G
H

C
3.4 Permuting an array

• Let's consider a different algorithm, based on choosing cards randomly

Pick a card, any card!

Repeat until all cards have been chosen
3.4 Permuting an array

• Let's consider a different algorithm, based on choosing cards randomly

___ ___ A
___ ___ B
___ ___ D
___ ___ E
___ ___ F
___ ___ G
___ ___ H

Pick a card, any card!

Repeat until all cards have been chosen

___ ___ A
___ ___ C
3.4 Permuting an array

• Let's consider a different algorithm, based on choosing cards randomly

_________ B
_________ D
_________ E
_________ F
_________ G
_________ H

Pick a card, any card!

Repeat until all cards have been chosen

_________ A
_________ C
3.4 Permuting an array

• Let's consider a different algorithm, based on choosing cards randomly

- Pick a card, any card!
- Repeat until all cards have been chosen

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th></th>
<th>D</th>
<th></th>
<th>E</th>
<th></th>
<th>F</th>
<th></th>
<th>G</th>
<th></th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th></th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.4 Permuting an array

- Let's consider a different algorithm, based on choosing cards randomly

Pick a card, any card!
Repeat until all cards have been chosen

__________ E
__________
__________ G
__________ D
__________ H
__________ B
__________ F
__________ A
__________ C

End of Oyster
3.5 Exercise

• Video intro
  – Let's implement our deck-shuffling algorithm

• Approach
  – Implement one part at a time
  – Print out your structures as you go
  – Introduce what is next (screenflow of following)

• To print an array named foo along with each index:

```java
for (int i=0; i < foo.length; ++i) {
    System.out.println("foo[" + i + "] is " + foo[i]);
}
```
3.5 Exercise

• **Video intro (continued)**
  
  – How do we choose a random integer between \( i \) such that \( 0 \leq i < n \) ?
    
    • We need such an index to find a value in an array of \( n \) elements
  
  – Recall that `Math.random()` returns the double \( r \) such that \( 0 \leq r < 1 \)
  
  – Then \( n \ast Math.random() \) returns a value \( v \) such that \( 0 \leq v < n \)
  
  – How do we convert that **double** value to an integer?
    
    • We use an `(int)` cast
  
  – **The code is then:**
    
    \[
    \text{int } i = (\text{int}) (n \ast \text{Math.random}());
    \]
• **Question card**
  
  – Declare a String array named `original` with contents "A", "B", ..., "H"
    
    • There should be 8 elements
    
    • Print out the array as shown
  
  – Declare a String array named `shuffled` to be the same size as `original`
  
  – Iterate backwards to fill in the `shuffled` array
    
    • Each time choosing a random element from `original`
    
    • Placing that random entry into the `shuffled` array
    
    • And then moving the cards up in the `original` array to close the gap
3.5 Exercise

- Video response
  - Go over the code for the solution
  - Highlight how the random index is computed
  - Highlight how the unused slot is filled in
    - And what happens if we do it the wrong way
  - Is it necessary to move the cards up?
    - Or could the last card be swapped with what was chosen?
3.6 Two-dimensional arrays

- A one dimensional array
  - A vector of data
  - Maps an int (index) to the array's type

- A two-dimensional array
  - A matrix of data
  - Maps an int pair (row, column) to the array's type

<table>
<thead>
<tr>
<th>&quot;A&quot;</th>
<th>&quot;poem&quot;</th>
<th>&quot;for&quot;</th>
<th>&quot;my&quot;</th>
<th>&quot;two&quot;</th>
<th>&quot;dimensional&quot;</th>
<th>&quot;array&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Each&quot;</td>
<td>&quot;line&quot;</td>
<td>&quot;has&quot;</td>
<td>&quot;similar&quot;</td>
<td>&quot;numbers&quot;</td>
<td>&quot;of&quot;</td>
<td>&quot;words&quot;</td>
</tr>
<tr>
<td>&quot;We&quot;</td>
<td>&quot;index&quot;</td>
<td>&quot;this&quot;</td>
<td>&quot;array&quot;</td>
<td>&quot;with&quot;</td>
<td>&quot;two&quot;</td>
<td>&quot;ints&quot;</td>
</tr>
<tr>
<td>&quot;One&quot;</td>
<td>&quot;each&quot;</td>
<td>&quot;for&quot;</td>
<td>&quot;the&quot;</td>
<td>&quot;row&quot;</td>
<td>&quot;and&quot;</td>
<td>&quot;column&quot;</td>
</tr>
</tbody>
</table>
3.6 Two-dimensional arrays

- A one dimensional array
  - A vector of data
  - Maps an int (index) to the array's type

- A two-dimensional array
  - A matrix of data
  - Maps an int pair (row, column) to the array's type

```java
String[][] words = {…};
```

<table>
<thead>
<tr>
<th>word</th>
<th>word</th>
<th>word</th>
<th>word</th>
<th>word</th>
<th>word</th>
<th>word</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;A&quot;</td>
<td>&quot;poem&quot;</td>
<td>&quot;for&quot;</td>
<td>&quot;my&quot;</td>
<td>&quot;two&quot;</td>
<td>&quot;dimensional&quot;</td>
<td>&quot;array&quot;</td>
</tr>
<tr>
<td>&quot;Each&quot;</td>
<td>&quot;line&quot;</td>
<td>&quot;has&quot;</td>
<td>&quot;similar&quot;</td>
<td>&quot;numbers&quot;</td>
<td>&quot;of&quot;</td>
<td>&quot;words&quot;</td>
</tr>
<tr>
<td>&quot;We&quot;</td>
<td>&quot;index&quot;</td>
<td>&quot;this&quot;</td>
<td>&quot;array&quot;</td>
<td>&quot;with&quot;</td>
<td>&quot;two&quot;</td>
<td>&quot;ints&quot;</td>
</tr>
<tr>
<td>&quot;One&quot;</td>
<td>&quot;each&quot;</td>
<td>&quot;for&quot;</td>
<td>&quot;the&quot;</td>
<td>&quot;row&quot;</td>
<td>&quot;and&quot;</td>
<td>&quot;column&quot;</td>
</tr>
</tbody>
</table>
3.6 Two-dimensional arrays

• A one dimensional array
  – A vector of data
  – Maps an int (index) to the array's type

• A two-dimensional array
  – A matrix of data
  – Maps an int pair (row, column) to the array's type

```
String[][] words = {…};
```

```
A
poem
for
my
two
dimensional
array
```

```
Each
line
has
similar
numbers
of
words
```

```
We
index
this
array
with
two
ints
```

```
One
each
for
the
row
and
column
```
3.6 Two-dimensional arrays

- A one dimensional array
  - A vector of data
  - Maps an int (index) to the array's type

- A two-dimensional array
  - A matrix of data
  - Maps an int pair (row, column) to the array's type

```java
String[][] words = {…};
```

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| "A"   | "poem" | "for"| "my"| "two"| "dimensional"| "array"
| "Each"| "line" | "has"| "similar"| "numbers"| "of"| "words"
| "We"  | "index"| "this"| "array"| "with"| "two"| "ints"
| "One" | "each" | "for"| "the"| "row"| "and"| "column"

"A"poem"for"my"two"dimensional"array"
"Each"line"has"similar"numbers"of"words"
"We"index"this"array"with"two"ints"
"One"each"for"the"row"and"column"
3.6 Two-dimensional arrays

<table>
<thead>
<tr>
<th>&quot;A&quot;</th>
<th>&quot;poem&quot;</th>
<th>&quot;for&quot;</th>
<th>&quot;my&quot;</th>
<th>&quot;two&quot;</th>
<th>&quot;dimensional&quot;</th>
<th>&quot;array&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Each&quot;</td>
<td>&quot;line&quot;</td>
<td>&quot;has&quot;</td>
<td>&quot;similar&quot;</td>
<td>&quot;numbers&quot;</td>
<td>&quot;of&quot;</td>
<td>&quot;words&quot;</td>
</tr>
<tr>
<td>&quot;We&quot;</td>
<td>&quot;index&quot;</td>
<td>&quot;this&quot;</td>
<td>&quot;array&quot;</td>
<td>&quot;with&quot;</td>
<td>&quot;two&quot;</td>
<td>&quot;ints&quot;</td>
</tr>
<tr>
<td>&quot;One&quot;</td>
<td>&quot;each&quot;</td>
<td>&quot;for&quot;</td>
<td>&quot;the&quot;</td>
<td>&quot;row&quot;</td>
<td>&quot;and&quot;</td>
<td>&quot;column&quot;</td>
</tr>
</tbody>
</table>

```
conceptually
```

```
actually
```
3.6 Two-dimensional arrays

Each line has similar numbers of words. We index this array with two ints: one each for the row and column.

<table>
<thead>
<tr>
<th>A</th>
<th>poem</th>
<th>for</th>
<th>my</th>
<th>two</th>
<th>dimensional</th>
<th>array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each</td>
<td>line</td>
<td>has</td>
<td>similar</td>
<td>numbers</td>
<td>of</td>
<td>words</td>
</tr>
<tr>
<td>We</td>
<td>index</td>
<td>this</td>
<td>array</td>
<td>with</td>
<td>two</td>
<td>ints</td>
</tr>
<tr>
<td>One</td>
<td>each</td>
<td>for</td>
<td>the</td>
<td>row</td>
<td>and</td>
<td>column</td>
</tr>
</tbody>
</table>

Conceptually, words is a one-dimensional array. Actually, each entry is itself another one-dimensional array.
3.6 Two-dimensional arrays

We index this array with two ints: one for the row and one for the column. Each line has similar numbers of words.

| "A" | "poem" | "for" | "my" | "two" | "dimensional" | "array" |
| "Each" | "line" | "bee" | "simile" | "numbers" | "of" | "words" |
| "We" | "index" | "this" | "array" | "with" | "two" | "ints" |
| "One" | "each" | "for" | "the" | "row" | "and" | "column" |

Conceptually, `words[2]` is the third entry of the `words` array. Actually, it is a one-dimensional array of 7 entries.
3.6 Two-dimensional arrays

<table>
<thead>
<tr>
<th>&quot;A&quot;</th>
<th>&quot;poem&quot;</th>
<th>&quot;for&quot;</th>
<th>&quot;my&quot;</th>
<th>&quot;two&quot;</th>
<th>&quot;dimensional&quot;</th>
<th>&quot;array&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Each&quot;</td>
<td>&quot;line&quot;</td>
<td>&quot;has&quot;</td>
<td>&quot;similar&quot;</td>
<td>&quot;numbers&quot;</td>
<td>&quot;of&quot;</td>
<td>&quot;words&quot;</td>
</tr>
<tr>
<td>&quot;We&quot;</td>
<td>&quot;index&quot;</td>
<td>&quot;this&quot;</td>
<td>&quot;array&quot;</td>
<td>&quot;with&quot;</td>
<td>&quot;two&quot;</td>
<td>&quot;ints&quot;</td>
</tr>
<tr>
<td>&quot;One&quot;</td>
<td>&quot;each&quot;</td>
<td>&quot;for&quot;</td>
<td>&quot;the&quot;</td>
<td>&quot;row&quot;</td>
<td>&quot;and&quot;</td>
<td>&quot;column&quot;</td>
</tr>
</tbody>
</table>

Each line has similar numbers of words.

We index this array with two ints.

Conceptually, words[2][6] is entry 6 of words[2].

Actually, it is a String, in this case the string "ints".
3.6 Two-dimensional arrays

Because of the way two-dimensional arrays are constructed, each row *could* have a different number of entries.
Because of the way two-dimensional arrays are constructed, each row *could* have a different number of entries.
3.6 Two-dimensional arrays

- Constructing by listing the elements

```java
String[][] words =
{
    {"A","poem","for","my","two","dimensional","array"},
    {"Each","line","has","similar","numbers","of","words"},
    { "We","index","this","array","with","two","ints"},
    { "One","each","for","the","row","and","column"}
};
```

<table>
<thead>
<tr>
<th></th>
<th>&quot;A&quot;</th>
<th>&quot;poem&quot;</th>
<th>&quot;for&quot;</th>
<th>&quot;my&quot;</th>
<th>&quot;two&quot;</th>
<th>&quot;dimensional&quot;</th>
<th>&quot;array&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Each&quot;</td>
<td>&quot;line&quot;</td>
<td>&quot;has&quot;</td>
<td>&quot;similar&quot;</td>
<td>&quot;numbers&quot;</td>
<td>&quot;of&quot;</td>
<td>&quot;words&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;We&quot;</td>
<td>&quot;index&quot;</td>
<td>&quot;this&quot;</td>
<td>&quot;array&quot;</td>
<td>&quot;with&quot;</td>
<td>&quot;two&quot;</td>
<td>&quot;ints&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;One&quot;</td>
<td>&quot;each&quot;</td>
<td>&quot;for&quot;</td>
<td>&quot;the&quot;</td>
<td>&quot;row&quot;</td>
<td>&quot;and&quot;</td>
<td>&quot;column&quot;</td>
<td></td>
</tr>
</tbody>
</table>
3.6 Two-dimensional arrays

• Constructing by listing the elements

```java
String[][] words = {
    {"A","poem"},
    {"Each","line","has","similar","numbers","of","words"},
    { "We","index","this","with","ints"},
    { "One","each","for","the","row","and","column"}
};
```

<table>
<thead>
<tr>
<th></th>
<th>&quot;A&quot;</th>
<th>&quot;poem&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Each&quot;</td>
<td>&quot;line&quot;</td>
<td>&quot;has&quot;</td>
</tr>
<tr>
<td>&quot;We&quot;</td>
<td>&quot;index&quot;</td>
<td>&quot;this&quot;</td>
</tr>
<tr>
<td>&quot;One&quot;</td>
<td>&quot;each&quot;</td>
<td>&quot;for&quot;</td>
</tr>
</tbody>
</table>
3.6 Two-dimensional arrays

• Constructing explicitly

```java
String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + i + "," + j + ")";
    }
}
```

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;(0,0)&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;(1,0)&quot;</td>
<td>&quot;(1,1)&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;(2,0)&quot;</td>
<td>&quot;(2,1)&quot;</td>
<td>&quot;(2,2)&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;(3,0)&quot;</td>
<td>&quot;(3,1)&quot;</td>
<td>&quot;(3,2)&quot;</td>
<td>&quot;(3,3)&quot;</td>
</tr>
</tbody>
</table>
3.6 Two-dimensional arrays

• Constructing explicitly

```java
String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + i + "," + j + ")";
    }
}
```

`words`
3.6 Two-dimensional arrays

• Constructing explicitly

```java
String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + i + "," + j + ")";
    }
}
```

```
words
```

```java
diagram
```
```
3.6 Two-dimensional arrays

- Constructing explicitly

```java
String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + j + "," + j + ")";
    }
}
```

```
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
3.6 Two-dimensional arrays

• Constructing explicitly

```java
String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + i + "," + j + ")";
    }
}
```

```plaintext
words

"(0,0)"
```
3.6 Two-dimensional arrays

• Constructing explicitly

```java
String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + i + "," + j + ")";
    }
}
```

```
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```
3.6 Two-dimensional arrays

- Constructing explicitly

```java
String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + i + "," + j + ")";
    }
}
```

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

```
words
```

```
(0,0)
(1,0)
(1,1)
```

114
3.6 Two-dimensional arrays

• Constructing explicitly

String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + i + "," + j + ")";
    }
}

words

"(0,0)"
"(1,0)"
"(1,1)"
"(2,0)"
"(2,1)"
"(2,2)"
"(2,3)"
"(3,0)"
"(3,1)"
"(3,2)"
"(3,3)"
3.6 Two-dimensional arrays

- Constructing explicitly

```java
String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + i + "," + j + ")";
    }
}
```

```
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
words
```

- Diagram of 2D array `words` with values:
  - (0,0)
  - (1,0)
  - (1,1)
  - (2,0)
  - (2,1)
3.6 Two-dimensional arrays

- Constructing explicitly

```java
String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + i + "," + j + ")";
    }
}
```

```
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
```
3.6 Two-dimensional arrays

• Constructing explicitly

```java
String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + i + "," + j + ")";
    }
}
```

```
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
```

```
words
```

```
"(0,0)"

```
"(1,0)"  "(1,1)"

```
"(2,0)"  "(2,1)"  "(2,2)"
```
3.6 Two-dimensional arrays

• Constructing explicitly

```java
String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + i + "," + j + ")";
    }
}
```

```
words

(0,0)
(1,0)
(1,1)
(2,0)
(2,1)
(2,2)
```

119
3.6 Two-dimensional arrays

• Constructing explicitly

```java
String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + i + "," + j + ")";
    }
}
```

```
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
```
3.6 Two-dimensional arrays

• Constructing explicitly

```java
String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + i + "," + j + ")";
    }
}
```

```
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
```
3.6 Two-dimensional arrays

- Constructing explicitly

```java
String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + i + "," + j + ")";
    }
}
```

```
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
```
3.6 Two-dimensional arrays

- Constructing explicitly

```java
String[][] words = new String[4][];
for (int i=0; i < words.length; ++i) {
    words[i] = new String[i+1];
    for (int j=0; j < words[i].length; ++j) {
        words[i][j] = "(" + i + "," + j + ")";
    }
}
```

```
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
```

End of Oyster
3.7 Exercise with 2-D arrays

• Video intro
  – Compute a table of products of integers
  – The element at table[i][j] is i * j

• Question card
  – Implement the table computation
  – Print the results so it looks like a table
    • Nicely formatted!

• Video response shows solution
  – Anticipates size of the results
3.8 Roundtable

• The birthday problem
  – N people walk into a room
  – How many of them have the same birthday

• Sketch out
  – How would we do this by hand
    • Turn to student (give instructions for what to do)
    • Live student
  – Representations
    • ith day of the year (tricky because not all months have same number of days)
    • month,day pair (easier)

• Solve and analyze
3.9 Conclusion

- Arrays allow us to create an indexed structure of related data
  - Temperature readings
  - Cards in a deck
- Arrays are a *mapping* from an index tuple
  \[ [3][4][9] \]
  to an element that is the type of the array
- Once allocated, an array cannot change size
  - `double t[5]` always has exactly 5 elements
- The elements are numbered from 0
  - It is a common mistake to think they start from 1
- Arrays can be rectangular or irregularly shaped
  - Rectangular is much more common
- Multidimensional arrays are really arrays of arrays of ...