2D Arrays

So far arrays have been 1-dimensional (lists).

2D Array - table, matrix

1D: int[] arr = new int[10];
2D: int[][] arr2d = new int[4][6]

Now, 2 indices:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#rows: arr2d.length
#cols: length of a row arr2d[0].length

To print:

for (int row = 0; row < arr2d.length; row++)
    for (int col = 0; col < arr2d[row].length; col++)
        print (arr2d[row][col]);
Survey Reporting: Reviewers rate B+J's flavors.

<table>
<thead>
<tr>
<th>F0</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
</tr>
</thead>
<tbody>
<tr>
<td>rev0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rev1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rev2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rev3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```java
double[][] ratings = new double[10][5];
```

(i) Avg rating per flavor?

Store in array:

```java
double [] avgs = new double[5];
```

```java
for (int col = 0; col < ratings[0].length; col++) {
    double sum(col = 0;
    for (int row = 0; row < ratings.length; row++)
        sum += ratings[row][col];
    avgs[col] = sum / avgs.length;
}
```

Running Time for n rows, m columns?

$\Rightarrow O(n \times m)$. If $n \approx m \Rightarrow O(n^2)$
Arrays

Looked at 1D, 2D arrays: size is fixed

Arrays - fixed-size list

ArrayList - dynamically resizeable list

- can hold only objects (not primitive data types)
- can hold different types of data (not good practice)

```
import java.util.ArrayList;

ArrayList<String> staff = new ArrayList<String>();
```

Important methods:

add (Object e) - inserts an element to end of list

add (int index, Object e) - at location index

get (int index) - returns element at location index

remove (int index) - removes & returns (list shrinks)

indexOf (Object e) - returns the index of the first occurrence of e or -1 if e not in the list

contains (Object e) - returns true if obj is in the list or false otherwise

size () - returns the # of elements in the list

ex:

```
staff.add("Michael")
staff.add("Jim")
staff.add("Dwight")
staff.remove(1)
```
```java
staff.add("Pam")

print(staff.indexOf("Dwight"))

staff.add(2, "Kelly")

print(staff.contains("Jim")) => false

staff.get(4) => error!

exit
```

acctsList is ArrayList of BankAccounts.

Deposit $100 to each account.

```java
for (int i = 0; i < acctsList.size(); i++)
    sum += acctsList.get(i).deposit(100);
```

```
0
    acctName =
    acctNum =
    balance = $9150
```

1

2

3
```
Array List implementation

implemented with a regular 1D array (internalArr) with initial size (ex 16).

when internalArr gets full:
- new array (tmpInternalArr) created with double the size (32)
- elements from internalArr copied over to tmpInternalArr

when internalArr gets half empty:
- new array tmpInternalArr with half the size created
- elements from internalArr copied over to tmpInternalArr

\[
\begin{array}{c}
\text{internalArray} \\
\text{\hspace{0.5cm}} \\mid \|	ext{P}\| \\
\end{array}
\Rightarrow
\begin{array}{c}
\text{tmpInternalArr} \\
\text{\hspace{0.5cm}} \| \| \| \| \| \| \| \| \|
\end{array}
\]
Running Time

Array List vs. (sufficiently large) array:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Array List</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Remove (from front)</td>
<td>O(n)</td>
<td>O(1) (set to -1, null)</td>
</tr>
<tr>
<td>Get</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>IndexOf</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Contains</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Size?</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

Maintain variable size to keep track of number of elements.

if add: do size++
remove: size--

ArrayLists also waste space!

Suppose internal array reaches capacity:

For next insert, internal array doubled:

If no more inserts: