Two-dimensional (2-d) arrays

- A table of values (references)
- Declare and access using *two* index values
- In Java, a 2-d array is an array of arrays (array of objects)
  - The orientation (row, column) is how *we choose to visualize* (organize) the table
  - By convention, we use *row-major* 2-d arrays

Multi-dimensional arrays

- Can have as many dimensions as you want
- Each dimension has its own constant **length**
- Since each dimension is an array of array references, it can have its own value of **length** ⇒ a *ragged* array

Creating a 2-d array

- Declare a reference \( x \) for a 2-d integer array:
- Create a 2-by-3 integer array \( y \):
- Create the following array:
  
  \[
  \begin{array}{ccc}
  2 & 4 & 6 \\
  8 & 1 & 3 \\
  \end{array}
  \]

Accessing a 2-d array

Given a reference \( x \) that points to a 2-d integer array. . .

- What is its height (# of rows)?
- What is \( x[0] \)?
- What is the length of the first row?
- How to access last element in the 2\(^{nd}\) row?
- How to access last element in last row?
Example 1: multiplication table and a useful pattern

/** = 2-d array for x rows of multiplication table */
public static int[][] multTable(int x)

Example 2: re-ordering rows

Given a 2-d int array m, re-order the rows such that the row with the highest row sum is the first row. Assume m is in row-majored order.

Thought question: what if you want to re-order the array such that the column with the highest column sum is the first column? How will the code fragment differ? What is the major difference?