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:
  
  2 4 6
  8 1 3

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 row 2?
- How to access last element in last row?
Example 1: 2-d array and a useful pattern

// Given a 2-d integer array \(x\), calculate the sum of all elements.
// Assume the array is rectangular.

What if the array is ragged instead of rectangular? Suppose all rows exist but the rows have different lengths.

What if not all rows exist and the existing rows have different lengths?

Example 2

Given a 2-d array \(m\), re-order the rows such that the row with the highest row sum is the first row.

Example 3

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?