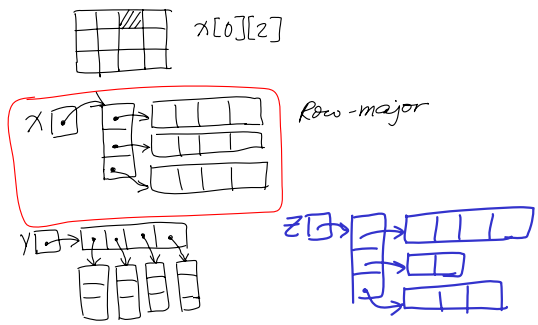
- Previous Lecture:
  - Polymorphism
  - `Object` class
  - `Abstract`
- Today's Lecture:
  - 2-d array
- Reading: Sec 6.4

2-d arrays

- A 1-d array is a *list* of values (references)
- A 2-d array is a *table* of values (references)
- 2-d array is referenced using two index values
- 2-d array in Java is really a 1-d array of 1-d arrays (i.e., an array of objects)
  - Orientation (row, column) is only how we choose to visualize the "table"
  - Convention: row-major

Multi-dimensional array

- Can have as many dimensions as you want
- A 2-d array is a 1-d array of 1-d arrays. Each 1-d array has its own constant **`length`** ⇒ you can have a ragged (not rectangular) 2-d array.



Creating a 2-d array

1. Declare a reference **`x`** for a 2-d integer array
2. Create a 2-by-3 integer array **`y`**
3. 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 **`int`** array. . .

1. What is its height (# of rows)?
2. What is **`x[0]`** ?
3. What is the length of the first row?
4. How to access last element in the second row?
5. How to access last element in last row?

Example 1

Given a 2-d integer array `x`, calculate the sum of all entries in the array. Assume the array is rectangular.

What if . . .

- The array is ragged instead of rectangular? Suppose all rows exist but the rows have different lengths.
- 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 2

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

```
//calculate row sums

//find index of row with max sum


//swap row of max sum with row 0
```

Example 3

Given a 2-d array `m`, re-order the columns such that the column with the highest column sum is the first column. Assume `m` is rectangular.

How will the code differ from the previous question?