2D Arrays

- Arrays can have two dimensions, useful for 2D data such as spreadsheets, images, tables, etc
- Here is a 3×3 array

```
// allocate and initialize array
int [][] a = {{2, 7, 6},
              {9, 5, 1},
              {4, 3, 8}};

// print array
for (int i = 0;  i < a.length;  i++)  {
  for (int j = 0;  j < a[i].length;  j++)
    print(a[i][j]+"␣␣");
  println();
  }
```

2D Arrays

- Technically, a 2D array is an array of arrays
- The first index of the array can be thought of as the row
- The second index of the array can be thought of as the column
- a.length is the number of rows
  (a is a 1D array of rows or type int[])
- a[0].length is the number of columns in row 0
  (a[0] is a 1D array)
- The array is stored row by row

Allocating Arrays

- A 2D array can be allocated as follows

```
int [][] a;          // a is an array
a = new int[7][];    // a has 7 rows
for (int i = 0;  i < a.length;  i++)
    a[i] = new int[12];
// each row has 12 elements
```

- the above array is 7×12 and contains 84 elements

- A 2D array can be also be allocated as follows

```
int [][] a = new int[7][12];    // a has 7
    rows and 12 columns
```

Ragged Arrays

- Not all rows in the array need to have the same number of elements
- Such an array is called a ragged array

```
// allocate and initialize array
int [][] a = {{1}, {1, 1}, {1, 2, 1},
      {1, 3, 3, 1}, {1, 4, 6, 4, 1}},
      {1, 5, 10, 10, 5, 1};

// print array
for (int i = 0;  i < a.length;  i++)  {
  for (int j = 0;  j < a[i].length;  j++)
    print(a[i][j]+"␣␣");
  println();
  }
```

- It is usually best to use the known size of the array as given by the `.length` value

Multidimensional Arrays

- Why stop at two dimensions?
- Arrays can have any number of dimensions!
- Here is an example of a 4D array

```
int [][][][] a = new int[10][10][10][10];
for (int i0 = 0; i0 < a.length; i0++)
 for (int i1 = 0; i1 < a[i0].length; i1++)
  for (int i2 = 0; i2 < a[i1].length; i2++)
   for (int i3 = 0; i3 < a[i2].length; i3++)
    a[i0][i1][i2][i3] = i1 + i2 + i3 + i4;
```

- Note this array requires $10 \cdot 10 \cdot 10 \cdot 10 = 10^4$ elements!
- To processes this array we use four nested loops
  - Outermost loop corresponds to leftmost index
  - Innermost loop corresponds to rightmost index