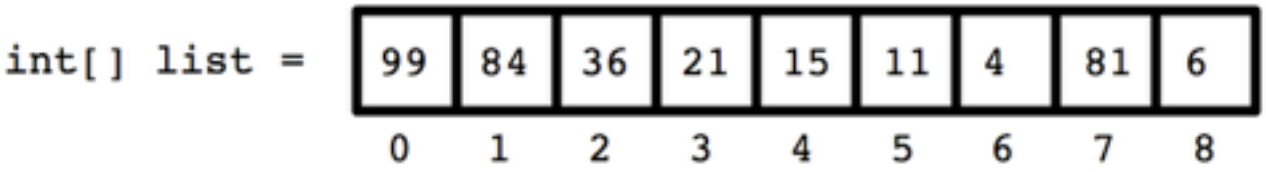
+

# Two Dimensional Arrays

# Two Dimensional Arrays

- So far we have studied how to store linear collections of data using a single dimensional array.




- However, the data associated with certain systems (a digital image, a board game, etc.) lives in two dimensions.
- To represent this data in our programs, we need a multi-dimensional data structure, that is, a multidimensional array.

# Two Dimensional Arrays

| | Chicago | Boston | New York |
|---|---|---|---|
| Chicago | 0 | 983 | 787 |
| Boston | 983 | 0 | 214 |
| New York | 787 | 214 | 0 |

```
int[][] distances = {
                     {0, 983, 787},
                     {983, 0, 214},
                     {787, 214, 0}
                    };
```

# Declaring Two Dimensional Arrays

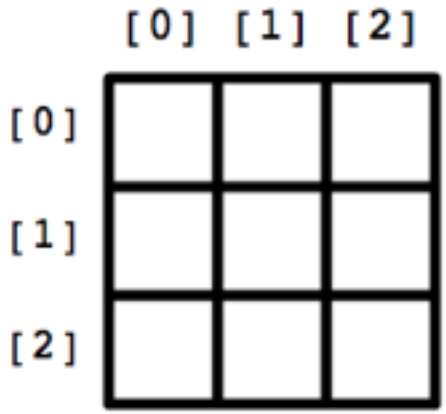
- You can declare a two dimensions array using almost the same syntax you would use to declare a single dimensional array. For example:

```
int[][] myList = new int[5][5];
```

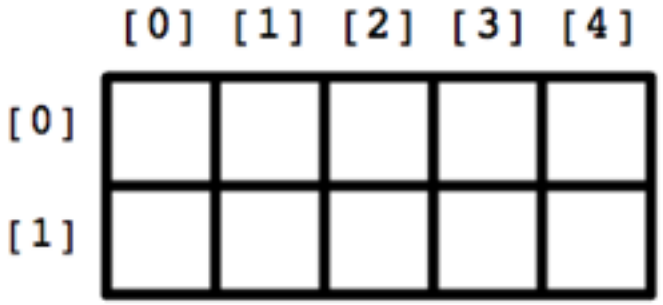
- This would create a 5 x 5 matrix of integers, all defaulted to a zero value upon creation. We generally think of the first number as the number of "rows" in your array and the second as the number of "columns"

# Declaring Two Dimensional Arrays

```
int[][] a = new int[3][3]
```




```
int[][] a = new int[2][5]
```



# Declaring Two Dimensional Arrays

- You can also create a two dimensional array with pre-specified values by using almost the same syntax as a single dimensional array. For example:

```
int[][] list = {
                  { 1, 2, 3 },
                  { 4, 5, 6 },
                  { 7, 8, 9 }
               };
```

- See *Creating2DArrays.java*

# Accessing Two Dimensional Arrays

- You can access the two dimensional array values using the same syntax as you would with a single dimensional array. For example, the first element in the first row is at position:

```
myList[0][0]
```

# Accessing Two Dimensional Arrays

```
int[][] list = new int[5][5];

list[0][0] = 50;

list[4][4] = 99;

list[2][1] = 11;
```

| | [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|---|
| [0] | 0 | 0 | 0 | 0 | 0 |
| [1] | 0 | 0 | 0 | 0 | 0 |
| [2] | 0 | 0 | 0 | 0 | 0 |
| [3] | 0 | 0 | 0 | 0 | 0 |
| [4] | 0 | 0 | 0 | 0 | 0 |

# Accessing Two Dimensional Arrays

```
int[][] list = new int[5][5];

list[0][0] = 50;

list[4][4] = 99;

list[2][1] = 11;
```

| | [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|---|
| [0] | 50 | 0 | 0 | 0 | 0 |
| [1] | 0 | 0 | 0 | 0 | 0 |
| [2] | 0 | 0 | 0 | 0 | 0 |
| [3] | 0 | 0 | 0 | 0 | 0 |
| [4] | 0 | 0 | 0 | 0 | 0 |

# Accessing Two Dimensional Arrays

```
int[][] list = new int[5][5];

list[0][0] = 50;

list[4][4] = 99;

list[2][1] = 11;
```

| | [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|---|
| [0] | 50 | 0 | 0 | 0 | 0 |
| [1] | 0 | 0 | 0 | 0 | 0 |
| [2] | 0 | 0 | 0 | 0 | 0 |
| [3] | 0 | 0 | 0 | 0 | 0 |
| [4] | 0 | 0 | 0 | 0 | 99 |

# Accessing Two Dimensional Arrays

```
int[][] list = new int[5][5];

list[0][0] = 50;

list[4][4] = 99;

list[2][1] = 11;
```

| | [0] | [1] | [2] | [3] | [4] |
|---|---|---|---|---|---|
| [0] | 50 | 0 | 0 | 0 | 0 |
| [1] | 0 | 0 | 0 | 0 | 0 |
| [2] | 0 | 11 | 0 | 0 | 0 |
| [3] | 0 | 0 | 0 | 0 | 0 |
| [4] | 0 | 0 | 0 | 0 | 99 |

- See *OffsetsIn2DArrays.java*

# Arrays of Arrays

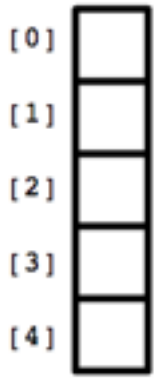
- Two dimensional arrays are really just one dimensional arrays that have been "chained" together. For example:

```
int[][] list = new int[5][5];
```

- Is a list of 5 elements. Each of those elements, however, references another single dimensional array, each of which is 5 elements long as well.
- Moreover, a two-dimensional array is really nothing more than an **array of arrays**.

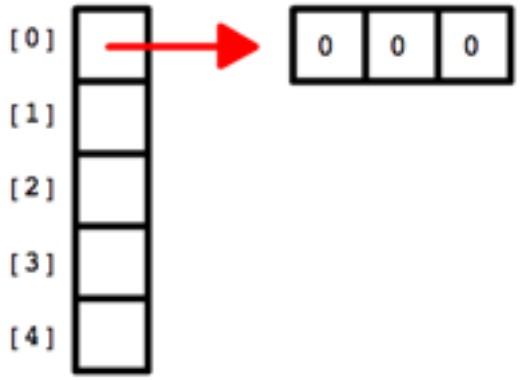
# Arrays of Arrays

```
int[][] list = new int[5][];

list[0] = new int[3];

list[1] = new int[3];

list[2] = new int[5];
```



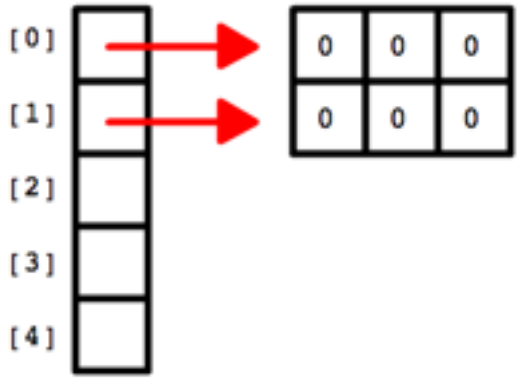
# Arrays of Arrays

```
int[][] list = new int[5][];

list[0] = new int[3];

list[1] = new int[3];

list[2] = new int[5];
```



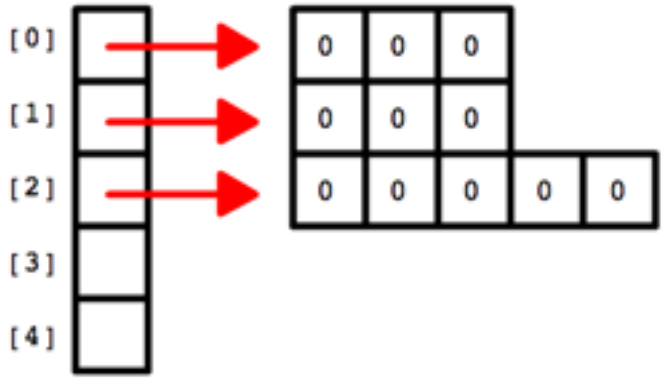
# Arrays of Arrays

```
int[][] list = new int[5][];

list[0] = new int[3];

list[1] = new int[3];

list[2] = new int[5];
```

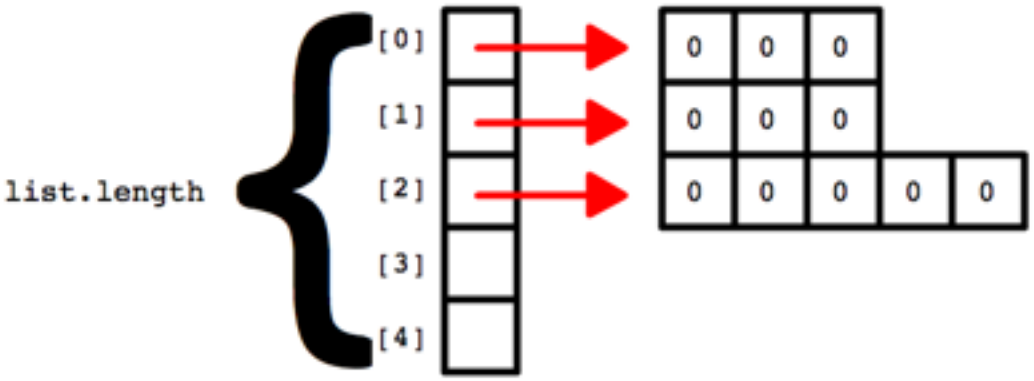


# Arrays of Arrays

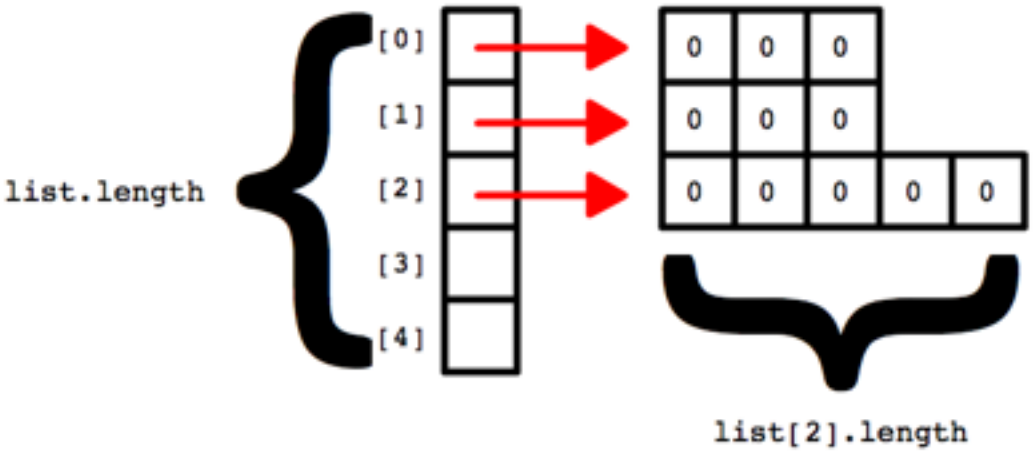
```
int[][] list = new int[5][];

list[0] = new int[3];

list[1] = new int[3];

list[2] = new int[5];
```



# Arrays of Arrays



# Arrays of Arrays



# Getting the Dimension Lengths

- The length of your "main" array in a 2 dimensional array can be obtained by referencing the following. We usually refer to this as the number of "rows" in the array

```
list.length
```

- The length of each sub-array in a two dimensional array can be obtained by referencing the following. We usually refer to this number as the "columns" in the array.

```
list[element].length
```

- Note that each row in a two dimensional array could have a different number of columns. We sometimes refer to these kinds of arrays as "ragged" arrays

# Looping Through 2D Arrays

- In order to iterate over all elements in a two dimensional array you will need to maintain two indexes – one for the row and one for the column

- This is usually done by setting up a nested for loop like this:

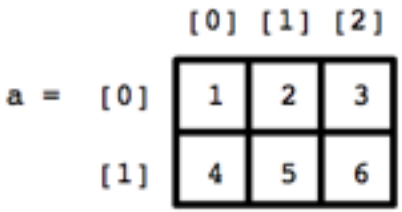
```
for (int row = 0; row < list.length; row++)
{
   for (int col = 0; col < list[row].length; row++)
      {
          // statements
          // list[row][col] = …
      }
}
```

# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```

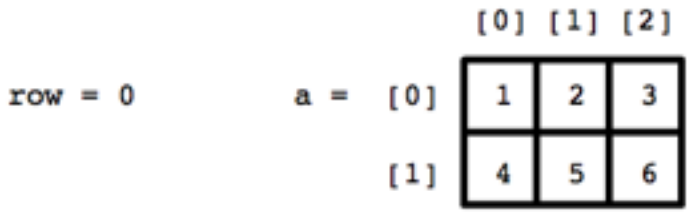
# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```



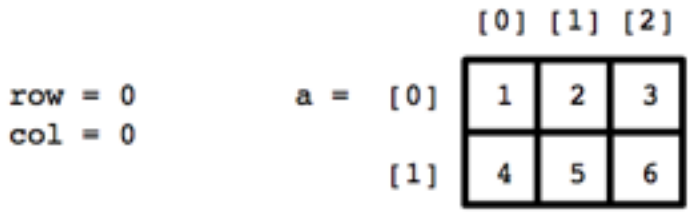
# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```



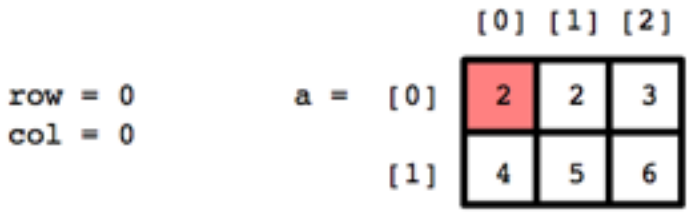
# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```



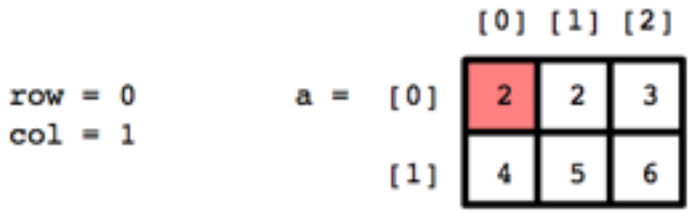
# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```



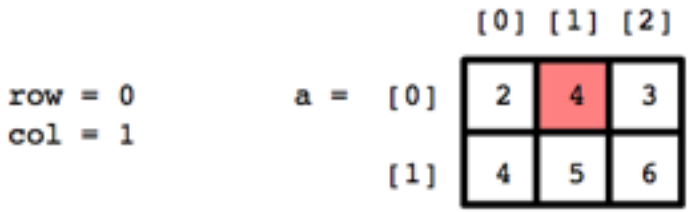
# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```



# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```



# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```

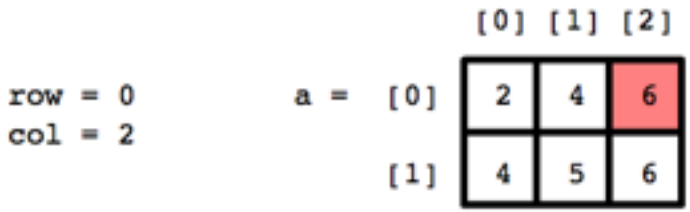
row = 0
col = 2

a =

| | [0] | [1] | [2] |
|---|---|---|---|
| [0] | 2 | 4 | 3 |
| [1] | 4 | 5 | 6 |

# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```



# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```

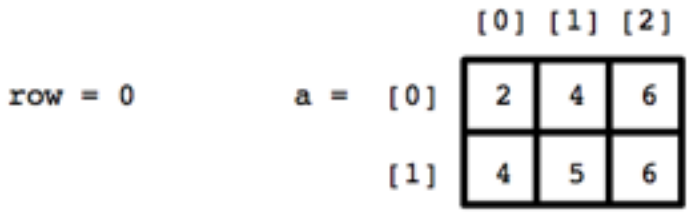
row = 0
col = 3

a =

| | [0] | [1] | [2] |
|---|---|---|---|
| [0] | 2 | 4 | 6 |
| [1] | 4 | 5 | 6 |

# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```



# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```

row = 1

a =

| | [0] | [1] | [2] |
|---|---|---|---|
| [0] | 2 | 4 | 6 |
| [1] | 4 | 5 | 6 |

# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```

row = 1
col = 0

a =

| | [0] | [1] | [2] |
|---|---|---|---|
| [0] | 2 | 4 | 6 |
| [1] | 4 | 5 | 6 |

# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```

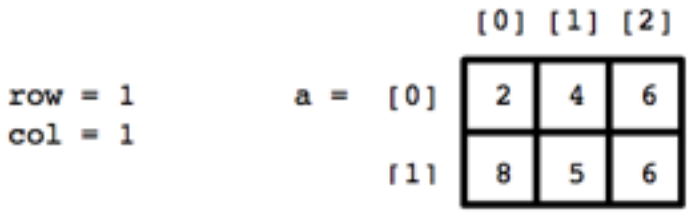
row = 1
col = 0

a =

| | [0] | [1] | [2] |
|---|---|---|---|
| [0] | 2 | 4 | 6 |
| [1] | 8 | 5 | 6 |

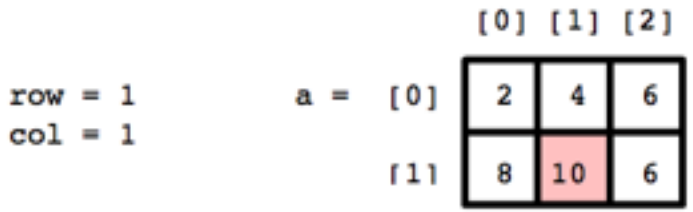
# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```



# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```



# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```

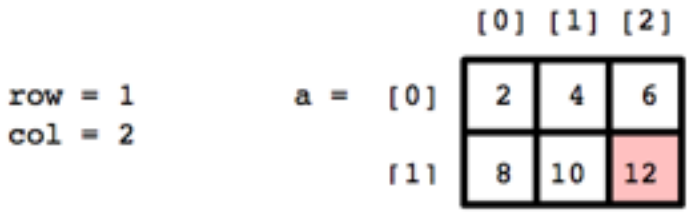
row = 1
col = 2

a =

| | [0] | [1] | [2] |
|---|---|---|---|
| [0] | 2 | 4 | 6 |
| [1] | 8 | 10 | 6 |

# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```



# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```

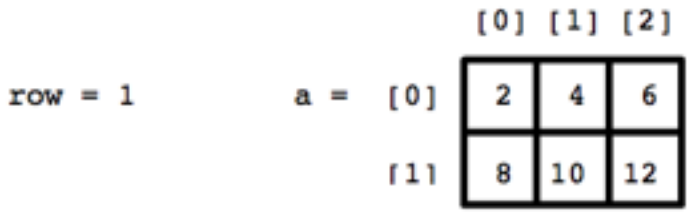
row = 1
col = 3

a =

| | [0] | [1] | [2] |
|---|---|---|---|
| [0] | 2 | 4 | 6 |
| [1] | 8 | 10 | 12 |

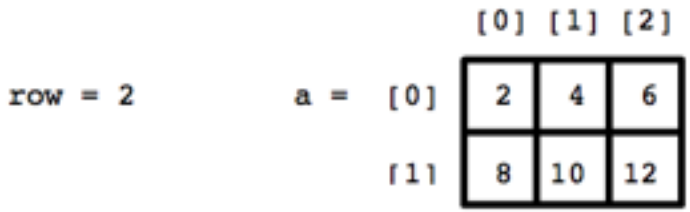
# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```



# Looping Through 2D Arrays

```
int[][] a = { {1,2,3},
              {4,5,6} };

for (int row = 0; row < a.length; row++)
{
  for (int col = 0; col < a[row].length; col++)
    {
       a[row][col] = a[row][col] * 2;
    }
}
```




- See *LoopingThrough2DArrays.java*

# Printing Two Dimensional Arrays

- Just as with single dimensional arrays, you cannot directly print a two dimensional array by using its variable name.

```
int[][] list = { {1,2,3}, {4,5,6} };
System.out.println(list);
// prints memory address
```

- Two dimensional arrays are reference types, so printing out the variable name associated with an array will only print out the memory address where the array is currently stored on the heap.

- Therefore you will need to set up a nested for loop in order to access and print each element in your array.

- See *Printing2DArrays.java*

# Summing Two Dimensional Arrays

- You can sum up all elements in an array by establishing a sum accumulator variable outside of the array and then accessing that variable as you visit each element. For example:

```
// set up a running sum
int sum = 0;

// iterate over the array
for (int row = 0; row < list.length; row++)
{
   for (int col = 0; col < list[row].length; col++)
      {
         sum += list[row][col];
      }
}
```

- See *Summing2DArrays.java*

# Programming Exercise

- Write the game Tic Tac Toe
- Players begin with an empty board of 9 cells
- Players can elect to place their token (X or O) in a cell. If the cell is not taken the token is assigned to that cell.
- Players take turns until either (a) all cells are occupied or (b) one player's token is found to occupy all three cells along any row, column or diagonal