2D Arrays

ACS-1904 LECTURE 2

What's this <String>?

- I asked you what the term was for the type definition of an array list in a previous class
- Any thoughts?

ArrayList<Cat> = new ArrayList<>();

What's this <String>?

- I asked you what the term was for the type definition of an array list in a previous class
- Any thoughts?

ArrayList<Cat> = new ArrayList<>();

Generics: add a type definition to an ArrayList for added type safety, the elimination of the need for casting, etc…

Arrays of arrays

- Arrays of 2 or more dimensions, (i.e. a 2D Array is like a table), in Java are arrays or arrays
- Example: times table
- Representation in memory
- Ragged arrays and partially filled arrays/tables
- Example: drivers & trips
- Example: working with matrices
- Addition and Multiplication

Arrays

- More than one value bound to an identifier
  - Each individual value is called an *element*
- Each element has an *index*. The *index* is the element's position within the array.
- Indexing begins at 0 (do you remember why?)
- And goes up to n -1, where n is the number of elements in the array.
- We can access each element of an array using a for-loop or a for-each-loop
- Arrays are homogenous, i.e. all elements are of the same data type, i.e. int

Arrays of arrays

- When the elements of an array are themselves arrays then we have multi-dimensional arrays, or, arrays of arrays
- E.g. Consider a table, *times*, that represents a times table:

times	**1**	**2**	**3**	**4**	**5**	**6**
1	1	2	3	4	5	6
2	2	4	6	8	10	12
3	3	6	9	12	15	18
4	4	8	12	16	20	24
5	5	10	15	20	25	30
6	6	12	18	24	30	36

Arrays of arrays

- The product, 4 x 6, is in the cell found at the intersection of the 4th row and 6th column

times	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	8	10	12
3	3	6	9	12	15	18
4	4	8	12	16	20	24
5	5	10	15	20	25	30
6	6	12	18	24	30	36

- As a 2-dimensional array we can access this element as

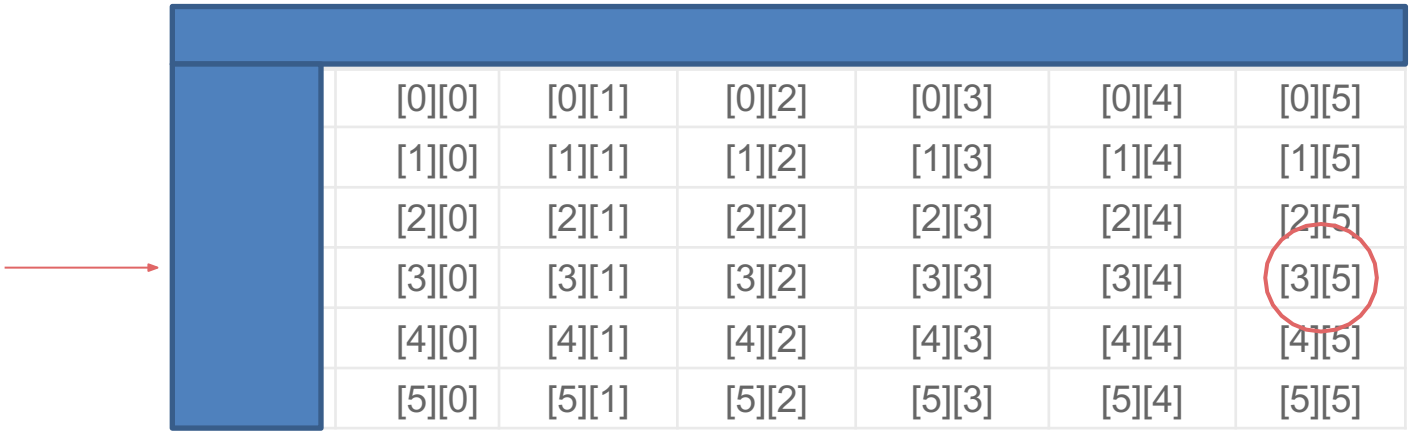
```
times[3][5]
```

sixth column

fourth row

Arrays of arrays

- If we disregard the row and column headings we can see how the 2D array is indexed



[0][0]	[0][1]	[0][2]	[0][3]	[0][4]	[0][5]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]	[1][5]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]	[2][5]
[3][0]	[3][1]	[3][2]	[3][3]	[3][4]	[3][5]
[4][0]	[4][1]	[4][2]	[4][3]	[4][4]	[4][5]
[5][0]	[5][1]	[5][2]	[5][3]	[5][4]	[5][5]

- As a 2-dimensional array we can access this element as

```
times[3][5]
```

sixth column

fourth row

Arrays of arrays (TimesTable.java)

We can declare this times table in Java as

```
int[][] times ={{1, 2, 3, 4, 5, 6},
                {2, 4, 6, 8, 10, 12},
                {3, 6, 9, 12, 15, 18},
                {4, 8, 12, 16, 20, 24},
                {5, 10, 15, 20, 25, 30},
                {6, 12, 18, 24, 30, 36} }
```

each element is an array

1st subscript is row.

2nd subscript is column, or equivalently, an index into the row.

- All values are enclosed in `{}`
- Each row is enclosed in `{}`
- All elements are comma-separated
- `times[3][5]` references value in the 4th row, 6th entry within the row.

Example(TimesTable.java)

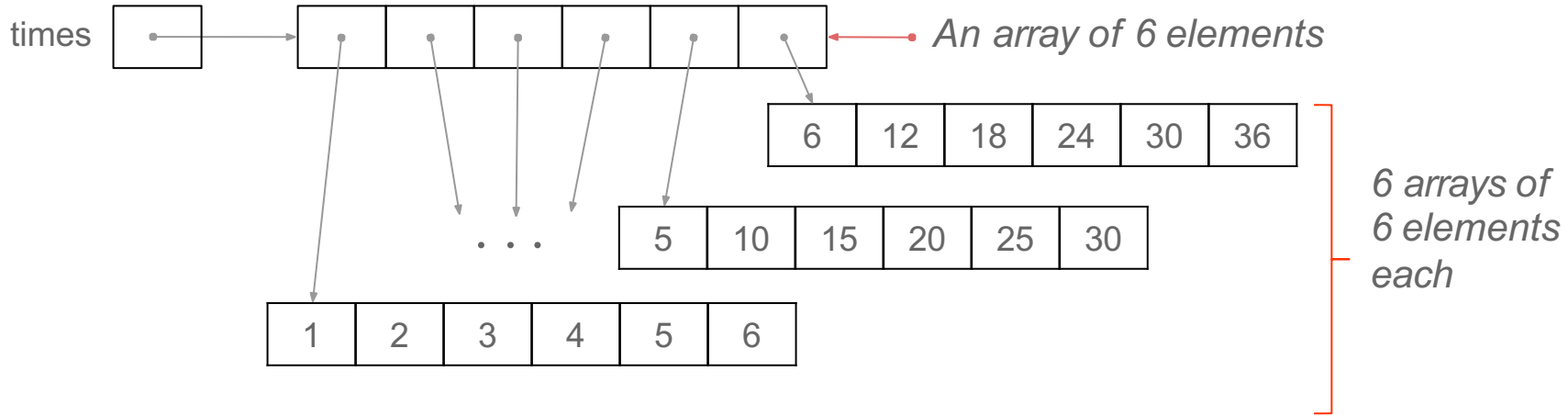
Consider the following code that prompts for 2 integers between 1 and 6 and reports their product:

```
int[][] times ={  {1, 2, 3, 4, 5, 6},
                  {2, 4, 6, 8, 10, 12},
                  {3, 6, 9, 12, 15, 18},
                  {4, 8, 12, 16, 20, 24},
                  {5, 10, 15, 20, 25, 30},
                  {6, 12, 18, 24, 30, 36 } };
Scanner kb = new Scanner(System.in);
System.out.println("enter two integers between 1 and 6");
int i = kb.nextInt();
int j = kb.nextInt();
int p = times[i-1][j-1];
System.out.println("The product of "+i+" and "+j+" is "+p);
```

Representation in memory

The JVM stores, in the memory location of the variable `times`, a reference to its elements (an array of …).

Each element has a reference to storage locations representing the column values that make up a row. Since these are just `int`s these values are stored in those locations.



Creating a 2D array(TimesTableMark2.java)

- As shown previously we can initialize an array of arrays.
- We can also declare the array and assign its values through calculations or input.
- It must first be declared and allocated, similar to a 1D array but now the declaration must explicitly state both the number of rows and the number of columns.
- Consider the code fragment on the next slide

Creating a 2D array(TimesTableMark2.java)

```
16      int[][] times = new int[6][6];
17
18      // load the array
19      for(int i = 0; i < 6; i++){
20          for(int j = 0; j < 6; j++){
21              times[i][j] = (i + 1) * (j + 1);
22          }// end for j
23      }// end for i
24
```

assignment statement

the element in the r^{th} row and c^{th} column is set to (r+1)(c+1)*

Creating a 2D array(TimesTableMark2.java)

```
16        int[][] times = new int[6][6];
17
18        // load the array
19        for(int i = 0; i < 6; i++){
20            for(int j = 0; j < 6; j++){
21                times[i][j] = (i + 1) * (j + 1);
22            }// end for j
23        }// end for i
24
```

Rather than an assignment statement the values could be assigned with a scanner.nextInt(), or by getting a value from a file

Snowfall.java

- Let's look at another example
  - This is a 2D-Array of monthly snowfall amounts for Denver
  - Note how the data is organized.
  - Each row is all of the data for one year
  - Each column is all of the data for one month over the span of years 2000-2014
- What if each year were stored in an array, so 2000 would be *year[0]*, 2001 would be *year[1]* and so on.
- Now we could have parallel arrays, *year* and *snowfall.*
- As an exercise change this code to use parallel arrays for year and snowfall. (note this type of parallel array implementation is also explored in exercise 5 at the end of chapter 2)

Ragged arrays

Java implements multidimensional arrays as arrays of arrays

Recall that an array has a length field specifying how many elements the array contains.

This means that each row can be a different length and each row has its own length field

Question: for a 10 x 10 times table, how many length fields are there?

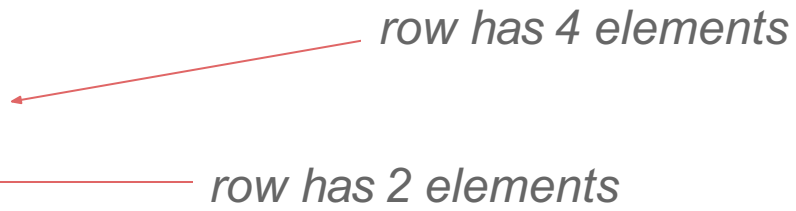
Consider Figure 2.4 which illustrates the km driven per trip for each driver

		Kilometres driven per trip				
Drivers	0	25	29	30	40	
	1	44	25			
	2	22	27	55	33	80
	3	55	57	45		
	4	31	42	49	46	

Ragged arrays (DriverTrips.java)

We can initialize the array

```
int[][] trips ={
        {25, 29, 30, 40},
        {44, 25},
        {22, 27, 55, 33, 80},
        {55, 57, 45},
        {31, 42, 49, 46}
};
```



Ragged arrays(DriverTrips.java)

We can display the trips for each driver

```
for (int i=0; i<trips.length; i++){
    System.out.print("driver: "+i+"\t");
    // number of trips for ith driver
    //    is trips[i].length
    for (int j=0;j<trips[i].length;j++){
        System.out.print(trips[i][j]+"\t");
    }
    System.out.println();
}
```

number of drivers

Size of i^{th} row ≡ number of trips for i^{th} driver

driver: row i

trip: column j

Ragged arrays

How many kilometers has each driver driven?

To calculate this we must calculate row sums.

```
for (int i=0; i<trips.length; i++){
    System.out.print("driver: "+i+"\t");
    // calculate row sum for driver
        int sum = 0;
    for (int j=0;j<trips[i].length;j++){
        sum += trips[i][j];
    }
    System.out.print(sum+"\n");
}
```

Row Wise Sorting

- Put the values in each row in some order, either ascending or descending
- Use the Arrays.sort() method on each successive row
  - Remember that in Java a 2D array is an array of arrays so each row is a 1d array

Row Wise Sorting

- Put the values in each row in some order, either ascending or descending
- Use the Arrays.sort() method on each successive row
  - Remember that in Java a 2D array is an array of arrays so each row is a 1d array
- Let's see how to use a static method to sort the rows of a table
  - And for good measure, we'll also look at using a static method to print a table.

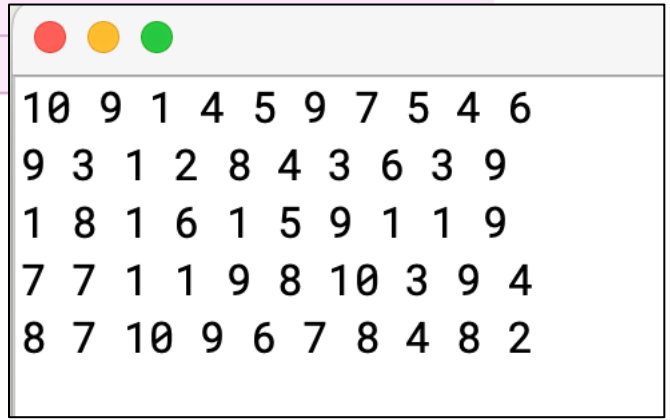
Row Wise Sorting

```
16
17      int table[][] = new int[ROWS][COLUMNS];
18
19      for(int i = 0; i < ROWS; i++){
20          for(int j = 0; j < COLUMNS; j++){
21              table[i][j] = r.nextInt(10) + 1;
22          }// end for j
23      }// end for i
24
```

- As a bit of review here we are loading the table with the assignment of random numbers

Row Wise Sorting

```
16
17      int table[][] = new int[ROWS][COLUMNS];
18
19      for(int i = 0; i < ROWS; i++){
20          for(int j = 0; j < COLUMNS; j++){
21              table[i][j] = r.nextInt(10) + 1;
22          }// end for j
23      }// end for i
24
```



- If we added some code to print the table it would look like this

Row Wise Sorting

```
36      // sort the table row wise
37      public static void sortTable(int[][] t, int size){
38          for(int i = 0; i < size; i++){
39              Arrays.sort(t[i]);
40          }// end i
41      }// end sort
42
```

- After calling the sortTable() method the output would look like this

```
BlueJ: Term
10 9 1 4 5 9 7 5 4 6
9 3 1 2 8 4 3 6 3 9
1 8 1 6 1 5 9 1 1 9
7 7 1 1 9 8 10 3 9 4
8 7 10 9 6 7 8 4 8 2

1 4 4 5 5 6 7 9 9 10
1 2 3 3 3 4 6 8 9 9
1 1 1 1 1 5 6 8 9 9
1 1 3 4 7 7 8 9 9 10
2 4 6 7 7 8 8 8 9 10
end of program
```

2D Arrays and Static Methods

```
43
44      public static void printTable(int[][] t, int r, int c){
45          for(int i = 0; i < r; i++){
46              for(int j = 0; j < c; j++){
47                  System.out.print(t[i][j] + " ");
48              }// end j
49              System.out.println();
50          }//end i
51      }// end printtable
52
```

- While we're here why not look at another common 2D array operation that can be decomposed into a static method

2D Arrays and Static Methods

- And calling the table static methods

```
25        printTable(table, table.length, table[0].length);
26        sortTable(table, table.length);
27
```

2D Arrays and Static Methods

- And calling the table static methods

```
25      printTable(table, table.length, table[0].length);
26      sortTable(table, table.length);
27
```

2D Array reference

Number of rows

Number of columns

2D Arrays and Static Methods

- And calling the table static methods

```
25    printTable(table, table.length, table[0].length);
26    sortTable(table, table.length);
27
```

2D Array reference

Number of rows

Matrices

A 2D matrix is a mathematical construct of rows and columns.

E.g.

$$\begin{bmatrix} 3 & 8 & 3 \\ 5 & 1 & 2 \end{bmatrix}$$

is a 2D matrix of 2 rows and 3 columns.

Sometimes called a 2 x 3 matrix

Matrices - scalar multiplication

Mathematical operations (add, subtract, scalar multiplication, transposition, dot product, etc.) are defined for matrices.

- Scalar multiplication
  - Consider the product *c* * *A* where *c* is a scalar value and *A* is a matrix, then *c* * *A* is computed by multiplying each element of *A* by the value *c*.

$$2 \begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 6 \\ 12 & 10 & 8 \end{bmatrix}$$

Consider the code below that computes $C = 2 * A$.

```
int[][] a ={
        {1, 2, 3},
        {6, 5, 4}};
int s = 2;
int [][] c = new int[2][3];
for (int i=0; i< a.length; i++)
        for (int j=0; j<a[i].length; j++)
            c[i][j] = s *a [i][j];
```

Matrix addition

To add two matrices of equal size we just add elements in corresponding positions.

$$\begin{bmatrix} 3 & 8 & 3 \\ 5 & 1 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 1 & 2 \\ 3 & 3 & 2 \end{bmatrix} = \begin{bmatrix} 4 & 9 & 5 \\ 8 & 4 & 4 \end{bmatrix}$$

If the matrices are not the same dimensions addition is not defined. i.e. they can't be added

(MatrixAddition.java

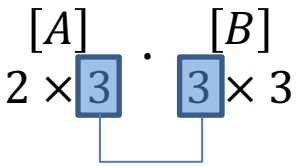
Adding 2 matrices:

```
// C = A + B
// For each c[i][j] in C
//    c[i][j] = a[i][j]+b[i][j]
// A and B must have the same
//    number of rows and columns

 for (int i=0; i< a.length; i++)
     for (int j=0; j<a[i].length; j++)
         c[i][j] = a[i][j] + b[i][j];
```

Matrix Multiplication a.k.a. dot product

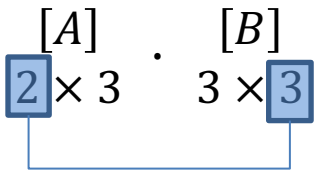
- Multiplying two Matrices is bit different.
- In order for the multiplication of two matrices to be defined (i.e. to be allowed to multiply them)….
- This operation is defined on two matrices if, for example, we have *[A]m x n and [B]n x o* the number of columns in A is equal to the number rows in B



This operation is allowed because A has 3 columns and B has 3 rows.

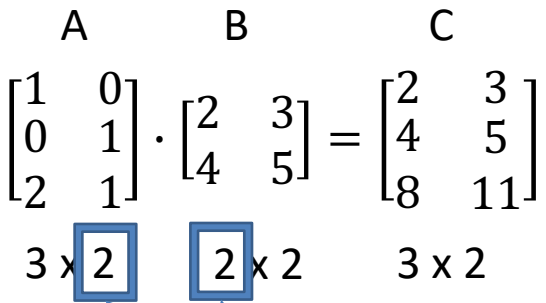
Matrix Multiplication a.k.a. dot product

- Multiplying two Matrices is bit different.
- In order for the multiplication of two matrices to be defined (i.e. to be allowed to multiply them)
- This operation is defined on two matrices if, for example, we have *[A]m x n and [B]n x o* the number of columns in A is equal to the number rows in B



The resulting matrix will be 2 x 3 since A has two rows and B has 3 columns

Matrix Multiplication a.k.a. dot product



We can multiply A * B because columnsA = rowsB

How does this work?

And the result matrix is 3 x 2 because A has 3 rows and B has 2 columns

Matrix Multiplication a.k.a. dot product

A B C

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \\ 8 & 11 \end{bmatrix}$$

$$= \begin{bmatrix} (A[0,0] * B[0,0] + A[0,1] * B[1,0]), & (A[0,0] * B[0,1] + A[0,1] * B[1,1]), \\ (A[1,0] * B[0,0] + A[1,1] * B[1,0]) & (A[1,0] * B[0,1] + A[1,1] * B[1,1]), \\ (A[2,0] * B[0,0] + A[2,1] * B[1,0]), & (A[2,0] * B[0,1] + A[2,1] * B[1,1]), \end{bmatrix}$$

Matrix Multiplication a.k.a. dot product

A B C

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \\ 8 & 11 \end{bmatrix}$$

$$= \begin{bmatrix} (A[0,0] * B[0,0] + A[0,1] * B[1,0]), & (A[0,0] * B[0,1] + A[0,1] * B[1,1]), \\ (A[1,0] * B[0,0] + A[1,1] * B[1,0]) & (A[1,0] * B[0,1] + A[1,1] * B[1,1]), \\ (A[2,0] * B[0,0] + A[2,1] * B[1,0]), & (A[2,0] * B[0,1] + A[2,1] * B[1,1]), \end{bmatrix}$$

Matrix Multiplication a.k.a. dot product

A B C

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \\ 8 & 11 \end{bmatrix}$$

$$= \begin{bmatrix} (A[0,0] * B[0,0] + A[0,1] * B[1,0]), & (A[0,0] * B[0,1] + A[0,1] * B[1,1]) \\ (A[1,0] * B[0,0] + A[1,1] * B[1,0]) & (A[1,0] * B[0,1] + A[1,1] * B[1,1]), \\ (A[2,0] * B[0,0] + A[2,1] * B[1,0]), & (A[2,0] * B[0,1] + A[2,1] * B[1,1]), \end{bmatrix}$$

Matrix Multiplication a.k.a. dot product

A B C

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \\ 8 & 11 \end{bmatrix}$$

$$= \begin{bmatrix} (A[0,0] * B[0,0] + A[0,1] * B[1,0]), & (A[0,0] * B[0,1] + A[0,1] * B[1,1]), \\ (A[1,0] * B[0,0] + A[1,1] * B[1,0]) & (A[1,0] * B[0,1] + A[1,1] * B[1,1]), \\ (A[2,0] * B[0,0] + A[2,1] * B[1,0]), & (A[2,0] * B[0,1] + A[2,1] * B[1,1]), \end{bmatrix}$$

Matrix Multiplication a.k.a. dot product

A B C

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \\ 8 & 11 \end{bmatrix}$$

$$= \begin{bmatrix} (A[0,0] * B[0,0] + A[0,1] * B[1,0]), & (A[0,0] * B[0,1] + A[0,1] * B[1,1]), \\ (A[1,0] * B[0,0] + A[1,1] * B[1,0]) & (A[1,0] * B[0,1] + A[1,1] * B[1,1]) \\ (A[2,0] * B[0,0] + A[2,1] * B[1,0]), & (A[2,0] * B[0,1] + A[2,1] * B[1,1]), \end{bmatrix}$$

Matrix Multiplication a.k.a. dot product

$$\begin{matrix} A & & B & & C \end{matrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \\ -8 & 11 \end{bmatrix}$$

$$= \begin{bmatrix} (A[0,0] * B[0,0] + A[0,1] * B[1,0]), & (A[0,0] * B[0,1] + A[0,1] * B[1,1]), \\ (A[1,0] * B[0,0] + A[1,1] * B[1,0]) & (A[1,0] * B[0,1] + A[1,1] * B[1,1]), \\ (A[2,0] * B[0,0] + A[2,1] * B[1,0]), & (A[2,0] * B[0,1] + A[2,1] * B[1,1]), \end{bmatrix}$$

Matrix Multiplication a.k.a. dot product

A B C

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \\ 8 & 11 \end{bmatrix}$$

$$= \begin{bmatrix} (A[0,0] * B[0,0] + A[0,1] * B[1,0]), & (A[0,0] * B[0,1] + A[0,1] * B[1,1]), \\ (A[1,0] * B[0,0] + A[1,1] * B[1,0]) & (A[1,0] * B[0,1] + A[1,1] * B[1,1]), \\ (A[2,0] * B[0,0] + A[2,1] * B[1,0]), & (A[2,0] * B[0,1] + A[2,1] * B[1,1]), \end{bmatrix}$$