Processing multiple arrays

```
public double[ ][ ] matrixAdd(double[ ][ ] a, double[ ][ ] b){
    int rows = a.length;
    int cols = a[0].length;

    if (b.length!=rows || b[0].length!=cols) { return null;  }

    double[ ][ ] ans = new double[rows][cols];
    for (int row=0; row<rows; row++){
        for (int col=0; col<cols; col++){
            ans[row][col] = a[row][col] + b[row][col];
        }
    }
    return ans;
}
```

$$\begin{pmatrix} 2 & 4 & 2 \\ 4 & 5 & 4 \\ 2 & 4 & 2 \end{pmatrix} + \begin{pmatrix} 6 & 3 & 2 \\ 5 & 1 & 4 \\ 7 & 3 & 0 \end{pmatrix} = \begin{pmatrix} 8 & 7 & 4 \\ 9 & 6 & 8 \\ 9 & 7 & 2 \end{pmatrix}$$

Processing multiple arrays

```
public double[ ][ ] matrixMultiply(double[ ][ ] a, double[ ][ ] b){
    int rows = a.length;     int cols = b[0].length;

    if (cols != b.length) { return null;  }

    double[ ][ ] ans = new double[rows][cols];
    for (int row=0; row<rows; row++){
        for (int col=0; col<cols; col++){
            for (int i=0; i<b.length; i++){
                ans[row][col] = ans[row][col]+a[row][ i ] * b[ i ][col];
            }
        }
    }
    return ans;
}
```

$$\begin{pmatrix} \boxed{2} & \boxed{4} & \boxed{2} \\ 4 & 5 & 4 \\ 2 & 4 & 2 \end{pmatrix} * \begin{pmatrix} 6 & 3 & 2 \\ 5 & 1 & 4 \\ 7 & 3 & 0 \end{pmatrix} = \begin{pmatrix} \boxed{12+20+14} & \boxed{6+4+6} & \boxed{4+16+0} \\ 24+25+28 & 12+5+12 & 8+20+0 \\ 12+4+14 & 6+4+6 & 4+16+0 \end{pmatrix}$$

Saving a 2D array to a file

- Write the grade table to a file:

A
B
B+
A-
B
B+
B-
C+
A
A-
B-
B+

Three people with 4 assignments each
or
Four people with 3 assignments each
or
Six people with 2 assignments each

A	B	B+	A-
B	B+	B-	C+
A	A-	B-	B+

A	B	B+
A-	B	B+
B-	C+	A
A-	B-	B+

A	B
B+	A-
A	B+
B-	C+
A	A-

- Writing the dimensions of the array first will help.

Saving a 2D array to a file

- Writing the dimensions in the file helps:

```
public void saveTable( String[][] grades, String fileName){
    try{
        PrintStream file =new PrintStream(new File(fileName));
        int rows = grades.length;
        int cols = grades[0].length;
        file.println(rows + " " + cols);
        for (int row=0; row<rows; row++){
            for (int col=0; col<cols; col++){
                    file.println(grades[row][col]);
            }
        }
    }catch(IOException e){UI.println("Table saving failed: "+e);}
}
```

Print number of rows and columns on first line of file.

Saving a 2D array to a file

- Alternate design:
  - assume file has been opened elsewhere and passed as argument
  - use "foreach" because not modifying the array.

```
public void saveTable( String[ ][ ] grades, PrintStream file){
    file.println(grades.length + " " + grades[0].length);
    for (String[ ] row : grades){
        for (String grd : row){
            file.println(grd);
        }
    }
}
```

- Note, you could pass System.out to the method to make it print to the terminal window!
  - (useful for debugging)

Loading 2D array from a file

- Assume first two tokens of file are the dimensions:

```
public String[ ][ ] loadTable( ){
    try {
        Scanner sc = new Scanner(new File(UIFileChooser.open()));
        int rows =sc.nextInt();
        int cols = sc.nextInt();
        String[ ][ ] ans = new String[ rows ][ cols ];
        for (int row=0; row<rows; row++){
            for (int col=0; col<cols; col++){
                    ans[row][col] = sc.next();
            }
        }
        return ans;
    } catch(IOException e){UI.out.println("Table loading failed: "+e);}
    return null;
}
```

Loading 2D array from a file

- Alternate, assuming
  - scanner is passed as argument
  - array is stored in a field

```
public void loadTable(Scanner sc ){
        this.dataArray = new String[ sc.nextInt() ] [ sc.nextInt() ];
        for (int row=0; row<this.dataArray.length; row++){
            for (int col=0; col<this.dataArray[row].length; col++){
                    this.dataArray[row][col] = sc.next();
            }
        }
}
```

Another file format for 2D arrays

- Suppose the array has only a few entries and many nulls
  - ie, a "sparse" array

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1			T1									
2					lects							
3												
4												
5	lects								lects			
6						Study						
7												
8												
9												
10						Exam						
11		end T3					T2					
12											end Ex	Grad
13												
14												
15												
16												
17					Grad					Study		
18				Break								
19												

Better to save just the non-null entries

File format

- size (months, days) [note: first and second index, not rows and columns!]
- month day entry

```
12  32
0  5  lects
1  11  end T3
2  1  T2
3  18  break
 :
 :
```

- To load:
  - read size and create calendar array
  - read month, day, entry, and assign entry to calendar[month][day]
- To save:
  - print size
  - for each non-null entry, print month, day, entry

Load sparse array

```
public void loadSparse(){
    try{
        Scanner sc = new Scanner( new File(UIFileChooser.open()));
        int months= sc.nextInt();
        int days = sc.nextInt();
        this.calendar = new String[months][days];
        while (sc.hasNext()){
            int month = sc.nextInt();
            int day = sc.nextInt()
            String entry = sc.next();
            this.calendar[month][dayl] = entry;
        }
        sc.close();
    }
    catch (IOException e){UI.println("Fail: " + e);}
}
```

Don't know how many entries there will be!

Save sparse array

```
public void saveSparse(){
    try{
        PrintStream ps = new PrintStream(new File(UIFileChooser.save()));
        ps.println(this.calendar.length+" "+ this.calendar[0].length);
        for (int month = 0 ; month < this.calendar.length; month++){
            for (int day = 0 ; day< this.calendar[month].length; day++){
                if (this.calendar[month][dayl] != null){
                    ps.println(month+" "+day+" "+this.calendar[month][dayl]);
                }
            }
        }
        sc.close();
    }
    catch (IOException e){UI.println("Fail: " + e);}
}
```