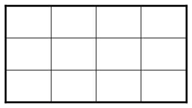
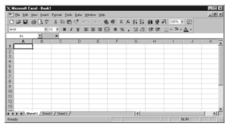
More About Arrays

2D arrays,
Command-line parameters

Two-Dimensional Arrays

- Two-dimensional arrays store data such that each data value is addressed using two subscripts.
- The typical visualization of the organization of the data in a 2D array is a matrix with rows and columns





Creating a 2D array

3

```
int[][] table = new int[3][4];
for (int row = 0; row < table.length; row++)
{
  for (int col = 0; col < table[row].length; col++)
  {
     table[row][col] = row * 10 + col;
  }
}
```

4

table	*0*	*1*	*2*	*3*
0	0	1	2	3
1	10	11	12	13
2	20	21	22	23

Example 1

- Create a 6 X 8 table of random numbers between 1 and 100 inclusive.

```
int[][] table = new int[____][_____];
for (int row = 0; row < table.length; row++)
{
  for (int col = 0; col < table[row].length; col++)
  {
     table[row][col] = ____________________________;
  }
}
```

Example 2

- Print the largest value in row 4.

```
int maxValue = _________________;
for (int col = 1; col < table[4].length; col++)
{
  if (_________________ > maxValue)

     maxValue = ________________________;
}
System.out.println(maxValue);
```

Example 3

- Print the average of column 2.

```
double sum = 0.0;
for (int row = 0; row < table.length; row++)
{
  sum += ______________________;
}
System.out.println(sum / table.length);
```

number of rows in table

Example 3 (w/ Ragged Arrays)
optional

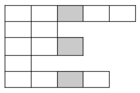
- Print the average of column 2.

```
int counter = 0;
double sum = 0.0;
for (int row = 0; row < table.length; row++) {

   if (___________________ >= 3) {

       sum += _______________________;
       counter++;
   }
}
if (counter > 0)
   System.out.println(sum / counter);
```

number of columns in current row



ragged array

Example 4

- Print the number of values less than 50 per row.

```
int counter;
for (int row = 0; row < table.length; row++)
{
   counter = _________;
   for (int col = 0; col < table[row].length; col++)
   {
      if (__________________________)
              counter++;
   }
   System.out.println("ROW " + row + ":" + counter);
}
```

A closer look at `main`

- Recall that `main` is a method.

```
public static void main(String[] args)
```

main is **`static`** because no objects need to be created to execute main

main's return type is **`void`** because main doesn't have a return statement (i.e. it doesn't return anything to another method)

main requires an **array of `String`** as its parameter (these are called **command-line parameters**)

Command Entry

- Instead of using Eclipse to compile and run programs, you can compile and run programs directly in a command window.
  - Mac: Terminal
  - Windows: Command Prompt (under Accessories)
- To compile a program:

  `javac` *classname*`.java`

  `javac *.java` (to compile all classes in a folder)
- To run a program:

  `java` *classname* (class that has `main`)

Command-Line Parameters

- When you run a program at the command line, you can supply parameters to `main` after the run instruction.

  Example: $\texttt{java FavoriteFood } \overbrace{\texttt{Tom pizza}}^{\text{array of String for main}}$

```
public class FavoriteFood {
  public static void main(String[] args) {
     System.out.println(args[0] +
          "'s favorite food is " + args[1]);
  }
}
```

Command-Line Parameters
with Numerical Values

- <u>All command-line parameters are strings</u>, even if they're meant to be numerical.

  Example Command: `java AverageComputer 78 84 95`

```
public class AverageComputer {
  public static void main(String[] args) {
     int sum = 0;
     for (int i = 0; i < args.length; i++)
          sum += Integer.parseInt(args[i]);
     System.out.println("Average = "
               + (double)sum/args.length);
  }
}
```

`Integer.parseInt` takes a `String` and returns an equivalent `int` value.