Outline

° Problem: How do store and manipulate data in tabular format

° Two-dimensional arrays – easy access with 2 indices

° This can get complicated
  • How many references are there to objects?

° Arrays as parameters
  • Arrays can be used as input and return values from methods
Two-Dimensional Arrays

- Declaration similar to one dimensional arrays
- Need to specify both the number of rows and columns during allocation
- Example:
  ```java
  final int COLS = 6, ROWS = 5;
  double[][] energyTable = new double[ROWS][COLS]
  ```

A one-dimensional array stores a list of elements
A two-dimensional array can be thought of as a table of elements, with rows and columns
Two-Dimensional Arrays

- To be precise, in Java a two-dimensional array is an array of arrays.
- A two-dimensional array is declared by specifying the size of each dimension separately:
  
  ```java
  int[][] scores = new int[12][50];
  ```
  
  You may interpret this as an array of 12 rows and 50 columns (short, fat...)
- A array element is referenced using two index values:
  
  ```java
  value = scores[3][6]
  ```
- The array stored in one row can be specified using one index (scores[3] is a 1d array.)

Arrays of Arrays

When we write

```java
energyTable = new double[ROWS][COLS];
```

This is shorthand for

```java
energyTable = new double[ROWS][];
for (int i = 0; i < ROWS; i++)
    energyTable[i] = new double[COLS];
```
Initializing Two-Dimensional Arrays

double[][] energyTable =
{
    {18.9, 19.4, 34.2, 3.9, 5.7, 0.3},
    {19.1, 19.3, 33.6, 3.0, 6.2, 0.2},
    {18.8, 19.6, 32.9, 3.1, 6.6, 0.2},
    {18.9, 20.3, 33.5, 2.8, 6.7, 0.2},
    {19.6, 20.8, 33.8, 3.1, 6.5, 0.2}
};

Populating energyTable

int y, s;
// reads 30 numbers needed to fill
// energyTable one row at a time
for (y = 0; y < ROWS; y++)
    for (s = 0; s < COLS; s++)
        energyTable[y][s] = in.readDouble();
### Computing Row Totals

```java
double[] yearTotals = new double[ROWS];

for (y = 0; y < ROWS; y++) {
    // compute total for year y
    yearTotals[y] = 0.0;
    for (s = 0; s < COLS; s++)
        yearTotals[y] = yearTotals[y] + energyTotal[y][s];
}
```

### Two-Dimensional Arrays

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>int[][]</td>
<td>2D array of integers, or array of integer arrays</td>
</tr>
<tr>
<td>table[5]</td>
<td>int[]</td>
<td>array of integers</td>
</tr>
<tr>
<td>table[5][12]</td>
<td>int</td>
<td>integer</td>
</tr>
</tbody>
</table>

This can be confusing. Look for the location of the brackets

2D array is really an array of 1D arrays
**Formatting Output**

- The `DecimalFormat` class can be used to format a floating point value in various ways.
- For example, you can specify that the number should be truncated to three decimal places.
- Constructor of the `DecimalFormat` class takes a string that represents a pattern for the formatted number.
  - `DecimalFormat fmt = new DecimalFormat("0.#");`
- Check this out in L&L 3.6

```java
public class TwoDArray
{
   //-------------------------------
   // Creates a 2D array of integers, fills it with increasing integer values, then prints them out.
   //-------------------------------
   public static void main (String[] args)
   {
      int[][] table = new int[5][10];

      // Load the table with values
      for (int row=0; row < table.length; row++)
         for (int col=0; col < table[row].length; col++)
            table[row][col] = row * 10 + col; // What is being loaded into table elements?

      // Print the table
      for (int row=0; row < table.length; row++)
         {
            for (int col=0; col < table[row].length; col++)
               System.out.print (table[row][col] + "t");
            System.out.println();
         }
   }
}
```

Make sure you totally understand what is going on here...
Two-Dimensional Arrays of Primitives

- Each row in a two-dimensional array is an array
- The rows can have different lengths
- Defining a primitive array where rows have the same length
  ```java
  int[][] data = new int[3][4];
  ```
- Defining a primitive data array where rows have different lengths (ragged array)
  ```java
  int[][] ragged = new int[2][];
  ragged[0] = new int[3];
  ragged[1] = new int[1];
  ```

Arrays Initialization Techniques

- Arrays of one-dimensional array
  - Declaring two-dimensional array `b[2][2]`
    ```java
    int b[][] = {{1, 2}, {3, 4}};
    ```
    - 1 and 2 initialize `b[0][0]` and `b[0][1]`
    - 3 and 4 initialize `b[1][0]` and `b[1][1]`
  - Declaring two-dimensional array `b[2][]`
    ```java
    int b[][] = {{1, 2}, {3, 4, 5}};
    ```
    - Row 0 contains elements 1 and 2
    - Row 1 contains elements 3, 4 and 5

- Two-dimensional arrays with rows of different lengths
  - Lengths of rows in array are not required to be the same
    - E.g., `int b[][] = {{1, 2}, {3, 4, 5}};`
Multidimensional Arrays

- An array can have many dimensions – if it has more than one dimension, it is called a **multidimensional array**.
- Each dimension subdivides the previous one into the specified number of elements.
- Each dimension has its own length constant.
- Because each dimension is an array of array references, the arrays within one dimension can be of different lengths.
  - these are sometimes called **ragged arrays**

Only responsible for 1D and 2D arrays in this course.

---

3D Arrays

[Diagram of a 3D array showing the first, second, and third dimensions with corresponding labels.]
Two dimensional array: Review

- There are 5 groups in one class
- There are 10 students in each group
- Their ages are stored in a two dimensional array `age[][]` array

Definition:
```
int [][] age = new int [5][10];
```

Two dimensional array

Notes:
- `age[0][0]` refers to 1st student’s age in 1st group
- `age[0][1]` refers to 2nd student’s age in 1st group
- ... ...
- `age[2][8]` refers to 9th student’s age in 3rd group
- ... ...
- `age[4][9]` refers to 10th student’s age in 5th group
Summary

° We can now make complicated data structures
  • Objects still the basic units of data storage

° Arrays are fundamental
  • Most data is stored in arrays
  • Allows for easy data access

° Two dimensional arrays allow us to store tables of data
  • In some cases the rows may be of different lengths

° Arrays of greater than two dimensions are possible but rarely used