Overview

- Chapter 7 in the textbook “Building Java Programs”, by Reges & Stepp.
- Array Basics and Terminology
- Why Arrays?
- Auto-Initialization and Initialization with Known Values
- Accessing Specific Array Elements
- Array Traversal
- Printing an Array
Array Basics and Terminology

- Computers and computer programs are very good at holding vast amounts of information (data). **Arrays** are better than individual variables at holding large amounts of data.

- **Array** - An indexed structure that holds multiple values of the same data type.

- **Index** – An integer indicating the position of a particular element in a data structure.

- **Element** – Each item within the array is called an element.

- **Zero-Based Indexing** - A numbering scheme used throughout Java (and many other languages) in which a sequence of values is indexed starting with 0 (element 0, element 1, element 2, and so on).
Why use arrays? Well, consider a program that must keep track of many temperatures, maybe two, three, or more...maybe dozens.

Your program could just declare all the needed temperature variables separately, like:

```java
double temperature1;
double temperature2;
double temperature3;
... 
```

But there has to be a better way, especially for many values?
... This is why!

- Instead of three (or more) different variables, you can declare one array that holds all the values your program needs:

  ```
  double[] temperatures = new double[3];
  ```

- The **syntax notation** for array declaration and sizing is as follows:

  ```
  <element type>[] <name> = new <element type>[<length>];
  ```
Is that all there is?

- Nope, there is a lot more to know, like:
  - What are the elements of an array automatically initialized to when the array is created?
  - How do you access individual elements of an array?
  - How do you traverse through an array in a loop within your program?
  - How do you print out an array, easily, say for debugging purposes?
Auto-Initialization

- **Auto-Initialization** – Depending on the data type of the array, its elements are automatically given a default, initial value when the array is created as follows based on type:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>0</td>
</tr>
<tr>
<td>double</td>
<td>0.0</td>
</tr>
<tr>
<td>char</td>
<td>‘\0’</td>
</tr>
<tr>
<td>boolean</td>
<td>false</td>
</tr>
<tr>
<td>objects</td>
<td>null</td>
</tr>
</tbody>
</table>
Initializing with Known Values

- You can also size an array and initialize it with known, specific values using the syntax that includes curly-braces and values separated by commas, as shown in the example below:

```java
int[] daysIn = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

- Where the `syntax notation` is as follows:

```java
<element type>[] <name> = {<value>, <value>, ..., <value>};
```
Accessing a Specific Element

- Any element of an array can be accessed using an index into the array.

- For example, to get the value associated with the third element of the `grades` array and use that value to set a new variable, `myGrade`, execute the following:

  ```
  int myGrade = grades[2];
  ```

- **IMPORTANT:** Note that the index of the third element is two (2). Remember, this is because of zero-based indexing.
Array Traversal

- Looping through an array and processing one, more or all of the elements is called “array traversal.”

- Arrays have a special **For** loop called the **For-Each** loop and it’s *syntax notation* is:

  ```java
  for (<type> <name> : <array>) {
      <statement>
      <statement>
      <statement>
      ...
  }
  ```
Example of For-Each

- An example:

```java
for (int x: temperatures) {
    if (x > average) {
        above++;
    }
}
```

- Notes:
  - The variable `x` is the loop variable, and it has to be declared with the same type as the array.
  - Each time through the loop, `x` has the value of the next element in the `temperatures` array.
  - The `average` and `above` variables were defined previously and shown for the purpose of the example.
Using a traditional **For** loop

- By using an integer loop variable as an index and the `.length` property of the array as an upper bounds, the traditional for loop may be utilized:

```java
for (int idx=0; idx < temperatures.length; idx++) {
    int temp = temperatures[idx];
    if (temp > average) {
        above++;
    }
}
```

- **Notes:**
  - `.length` is a property of an array, not a method (no parameters).
  - The index of the last element is one less than the length (due to **zero-based indexing**).
Printing an Array

- Attempting to print an array directly does not work. The following will output gibberish:

```
System.out.println(temperatures);  // BAD
```

- Instead, an array can be printed, one element at a time using the array traversal algorithms (for-each and for) as shown before and printing each element within the loop, or...

- The `Arrays` class (note the uppercase) can be utilized along with its `.toString()` method to return all the values of an array as a formatted String ready for printing, as in the following example:

```
System.out.println(Arrays.toString(temperatures));
```
More on Arrays...

- More on arrays in future lectures...