What is an Array?

- An array is a collection of variables.
- Arrays have three important properties:
  - arrays represent a group of related data (for example, temperature for the last five days, or stock prices for the last 30 days.)
  - all data within a single array must share the same data type (for example, you can create an array of ints or an array of floats, but you cannot mix and match ints with floats.)
  - The size of an array is fixed once it is created.

What exactly are we talking about?!

- Let’s say you have ten students and you want to save the grades for each of them for use throughout your program (i.e. we need to remember every grade not just loop and count them or average them, etc.)
- Could do it the hard way:
  - Set up ten variables called studentOneGrade, studentTwoGrade, studentThreeGrade, etc.
  - Very difficult to use and manipulate
  - Instead, could use an array to do it the easy way…

Using an array variable

- Create an array variable called studentGrades[10]
- Declared as follows:
  ```java
  int studentGrades[] = new int[10];
  ```
- This sets up a location in memory for 10 integers which can be referenced using studentGrades[ # ] where # is the particular student you want to look at.
Array Naming Considerations

• The rules for naming variables apply when selecting array variable names
  – Composed of letters, digits, dollar signs and underscore characters
  – Cannot start with a digit
• Follow all the good programming hints recommended for variable names
  – i.e. just think of it as an ordinary variable when making up the name
    • In addition it is bad style to end an array name with a digit
      – i.e. array3[5]

Parts of an Array

Parts of the array

• The array has some terminology we haven’t seen yet
  – Elements
  – Index
    • Position Number
    • Subscript
  – Zeroth element
  – Keyword “new”

Elements

• Refers to the individual items represented by the array. For example,
  – an array of 10 integers is said to have 10 elements
  – an array of 15 floats has 15 elements
  – an array of 5 characters has 5 elements
  – and so on…

Index (Position Number or Subscript)

• Refers to one particular element in the array
• Also known as position number or, more formally, as a subscript
• The first element in an array is represented by an index or subscript of 0 (zero). For example,
  – studentGrades[ 0 ]
• This first position is known as the zeroth element
• The second position is referred to by
  – studentGrades[ 1 ]

Figuring out the array positions

• In Java, an arrays’ elements start out at index 0 and go up to (the number of elements – 1)
• For example, our array of 10 student grades filled in with grades might look like:

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>85</td>
<td>76</td>
<td>99</td>
<td>38</td>
<td>78</td>
<td>98</td>
<td>89</td>
<td>90</td>
<td>82</td>
<td>88</td>
</tr>
</tbody>
</table>
Array positions (cont’d)

• We can access them by specifying the array name followed by square brackets with the index number between them. For example,
  
  System.out.println("The third student’s grade is "+studentGrades[2]);

• Would print only the third integer spot in the array (remember 0 is the first, 1 is the second, and 2 is the third element’s index / subscript)

• The output would look like the following:
  The third student’s grade is 99

• The index scheme starting at 0 may be initially confusing.
• This is the cause of many off-by-one errors so study the concept carefully
• The element in the array’s first position is sometimes referred to as the zeroth element.
• Notice the difference between “position” and “element number”

Keyword new

• As we will see, the keyword new is used in Java when you wish to create a new object.
• In Java, arrays are objects.
• As with all data members of objects, the elements of each position in a new array will automatically be initialized to the default value for the array’s type.

Some powerful features of arrays

• Can use expressions as the subscript
  – E.g. if variables a = 1 and b = 2
  – studentGrades[a + b] would be the same as writing studentGrades[3]

• Can use array elements in expressions
  – E.g.
  
  int gradeTotal = 0;
  gradeTotal = studentGrades[0] +
  studentGrades[1] +
  studentGrades[2] +…etc…
  studentGrades[9];
  – Would add up all the array elements and store them in gradeTotal.

Powerful features (cont’d)

• Can set array elements equal to other values or expressions. For example,
  
  studentGrades[1] = 100;

• This would set the array element with a subscript of 1 to a grade of 100. That is, the second student would have a grade of 100.

• Java allows us to access an array’s length by using the following notation:
  – studentGrades.length would evaluate to 10
Using Arrays

So how do we use arrays?

- Same concepts as other variables apply
  - Must declare the array
  - Must initialize the array (unlike regular variables, Java will automatically initialize arrays with default values)
- Arrays variables are actually reference variables.
  - Can use arrays in expressions and methods, setting elements’ values or using their values, similar to the use of ordinary variables

Declaring an array

- First you can declare an array reference variable (you must specify the type of the elements in the array):
  - int [] myFirstArray;    //declares an array
    //variable for ints
  - Note: the line above does not allocate memory for the array (it cannot since we have not said the size of the array yet). It only sets aside enough memory to reference the array. Until we create an array, the reference is to null.
- Next, we create the array and “point” the reference to it:
  - myFirstArray = new int [10];
  - To create the array, we need the number of elements in the array so the computer can set aside adequate memory for the array.
- We can combine the declaration and allocation lines into one line as follows:
  - int [] myFirstArray = new int [10];

Initializing array with a for loop

- After declaring an array, you can initialize it in the body of your program by using a for loop:
  - int [] myFirstArray = new int [ 5 ];
  - for (int i = 0 ; i <= 4 ; i++ )
    - myFirstArray[i ] = i ;
  - // end for
  - Note the upper bound is 4, not 5! That is, you loop through 0 to 4 to initialize an array with 5 elements
  - Note: Array elements are initialized with default values. Numeric types get 0, boolean get false and char gets the null character (ascii code 0).

Declaring arrays (cont)

- You can use a constant to set the size of the array
  - final int NUM_STUDENTS_IN_CLASS = 8
    - int [] exams = new int [NUM_STUDENTS_IN_CLASS ];

- In Java, you do not need to know the size of the array at COMPILE time. Instead you can know the size at RUN time. For example, the following is legal:
  - inputString = JOptionPane.showInputDialog("How many students ?");
  - int students = Integer.parseInt (inputString);
    - int [] myFirstArray = new int [students];

Initializing an Array

- You can initialize an array when you declare it, as you do with other variables
- Syntax is slightly different, as you are now initializing more than one element at a time
- One way at declaration, using initializers:
  - int myFirstArray[ ] = { 0, 0, 0, 0, 0 };

- Note the braces around the initial zeroes which themselves are separated by commas.
- Also note that creating arrays in this way eliminates the need for the keyword new.
Accessing elements with for loop

- Can use a for loop to print the contents of an array

```java
int [] myFirstArray = new int [ 5 ];
for (int i = 0 ; i <= myFirstArray.length - 1; i++ )
{
    System.out.println ("array element "+ i + " is equal to "+ myFirstArray [i]);
} // end for
```
- Note the use of `myFirstArray.length` instead of using the size itself. It is better style to do so. Why?

Array Bounds

- Very Important: Java does not provide array bounds checking at compile time.
- This means that your program will crash at run time if you try to access memory outside of an array’s bounds.
- For example, suppose you have:

```java
int [] myFirstArray = new int [ 10 ];
myFirstArray[100] = 45;
```
- 100 is very far outside your defined size (0..9)
- However, the compiler will not indicate an error.

References and Reference Parameters

- Two ways to pass arguments to methods
  - Pass-by-value
    - Copy of argument’s value is passed to called method
  - In Java, every primitive is pass-by-value
  - Pass-by-reference
    - Caller gives called method direct access to caller’s data
    - Called method can manipulate this data
    - Improved performance over pass-by-value
  - In Java, every object is simulated pass-by-reference
    - Technically, we are using pass by value but the "value" we are passing is a reference to the array.

Passing Arrays to Methods

- To pass array argument to a method
  - Specify array name without brackets
    - `Array hourlyTemperatures` is declared as
    ```java
    int hourlyTemperatures = new int [ 24 ];
    ```
  - The method call
    ```java
    modifyArray( hourlyTemperatures );
    ```
  - Passes array `hourlyTemperatures` to method `modifyArray`

Returning arrays from methods

- We have seen examples of methods that modify an array. Since the array is passed by reference, it is modified in the original method as well.
- Another way to return information in an array is to explicitly return the array.
- For example, a method with the header:

```java
public static int [] makeArray ()
```
- Returns an integer array.
Returning arrays (cont)

- In order to return an array, the method creates the array first.

```java
public static int[] makeArray ()
{
    int[] myArray = new int[10];
    for (int i; i < myArray.length; i++)
        array[i] = i;
    return myArray;
}
```

- The method above creates a new array called myArray, initializes each element and returns the reference to myArray.

Copying Arrays (Liang)

In this example, you will see that a simple assignment cannot copy arrays in the following program. The program simply creates two arrays and attempts to copy one to the other, using an assignment statement.

![Diagram showing before and after assignment](image)

Using a loop:

```java
int[] sourceArray = {2, 3, 4, 5, 10};
int[] targetArray = new int[sourceArray.length];
for (int i = 0; i < sourceArrays.length; i++)
targetArray[i] = sourceArray[i];
```

The `arraycopy` Utility (Liang)

```java
arraycopy(sourceArray, src_pos, targetArray, tar_pos, length);
```

Example:

```java
System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);
```