Java Arrays

- Single Dimensional Arrays
- Arrays as Objects
- Multi Dimensional Arrays

Multiple Values

- Suppose you have a program that calculates your grade average over three courses
  - You could use three variables, grade1, grade2, and grade3 and perform calculations
  - If you want to average in a fourth course, you add another variable declaration and change code to also use new variable
- This works fine for your first quarter, but what about by the time of your senior year?
  - Having a separate variable for every single course would be tedious, error prone, and hard to understand
  - We would like to deal with something like a list of grade values
  - We would like to use a loop over the list, e.g., for each grade in the list, add it to the running sum then divide by the number of grades in the list
What we need

- What are the characteristics of this problem?
  - Many values, no particular limit to how many
  - All the values are the same type
    - E.g., they are all grades
  - We want to deal with the values symbolically
    - We do not want a separate hard coded name for each value, but rather want to be able to deal with them as the first, second, third, etc.
    - Even better, we would like to be able to use a counting variable to access these data values

- Solution: Java Arrays

Java Arrays

- Use Java arrays to manage large data sets
- Arrays are like indexed lists of primitive or object values of the same type
  - E.g., a bunch of integers, or a bunch of Strings, or a bunch of BankAccount objects
- Only one variable name (the name of the array)
  - The element operation allows access to each individual value in the array
- Loops are typically used for array processing
  - Need to be able to know how many values are in the array
  - Just as loops are more expressive than if-else statements for large computations, arrays are more expressive than single variables
    - Easier to exploit computational power
More precisely...

- **An array** is an ordered list of values of the same type

  ![Array Example]

- In Java, an array of size N is indexed from zero to N-1
- The array pictured has 10 values, and the indices range from 0 to 9

Java Array Syntax

- Declaring array variables

  ```java
doouble [] rainfall;    // inches of rain
String [] grades;        // Letter grades
BigInteger [] pageHits;  // Web page hits
```

- An array is an **object**, not a primitive type
  - Regardless of the element type
- Declaration produces a single object variable
  - Like any object variable, it is **null** until initialized
**Java Array Syntax**

- Initializing an array variable
  
  ```java
  double[] rainfall = new double[12];
  grades = new String[10];
  pageHits = new BigInteger[7];
  ```

- The array creation must specify the size of the array
  - I.e., the number of elements in it
  - Like defining that many variables of the element type
- Array of objects does not create the objects
  - Only the array itself
  - Like having a bunch of object variables, all initialized to null

**Size of an Array**

- The size of an array is determined when it is created
  - Arrays are of fixed size
  - The size cannot change (can not grow an array)
- Get the size of array from the `length` data member
  - `length` is an unsigned int value
  - It is a public data field - not a method of the array object

```java
for (int i = 0; i < pageHits.length; i++)
```
Accessing Elements of an Array

- Access an element of an array using the array operator `[]` and an integer index.

  ```java
  rainfall[0] = 7.65;
  System.out.println(rainfall[0] + " in Jan");
  ```

  Store each input grade string
  ```java
  grades[i] = scan.next();
  ```

  Store i\textsuperscript{th} hit count
  ```java
  pageHits[i] = new BigInteger(scan.next());
  ```

- The index must be within range of the array size
- Otherwise an ArrayIndexOutOfBoundsException exception occurs

 ArrayTest.java

---

More Array Initialization

- An array is often used as a table
  - Values could be hard coded rather than determined at run time
  - Size of array is automatically determined by compiler
  - Braces enclose comma separated list of initial values

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

  mdays.length is 12

  ```java
  String [] grades = {"A", "B", "C", "D", "F"};
  ```

  grades.length is 5

  ```java
  BigInteger [] primes = {
  new BigInteger("689572171629632424814677540353"),
  new BigInteger("93746241389848601401237229733")
  };
  ```

  primes.length is 2
Java Array Summary

- Arrays are for homogeneous ordered lists
  - All objects are the same type
- Array element indexing begins with zero
  - Automatic bounds checking: 0 to length-1
- Arrays are objects
  - Must be created with new
- Creating an array of objects does not create the objects in the array, just space for references
  - Unless an initializer list is given
  - Creation of an array of a primitive type does create space for the elements (with or without an initializer list)
- Arrays are constant in size
  - The size is available through the public member length
- [] is the array operator
  - Used in the syntax of defining an array type
  - Used to access elements for reading or writing

Foreach Loops

- Special form of a for loop
  - Works with arrays
  - No need for an index value or to know length of array
  - Idea is to iterate over all the element values in the array, starting with the first element

```java
String[] names = {"Larry", "Moe", "Curly", "Shemp"};
for (String stooge : names) {
    System.out.println(stooge + " was one");
}
```
Passing Arrays to Methods

- Arrays can be passed to methods
  - No need to also pass the length since the array object "knows" its own length
- An array is an object, so method gets a reference
  - No copy is made of the array
  - Arrays are mutable, so method might change the array values

```java
main method
int[] mdays = {31, 28, ...}
double a = avgMonth(mdays);

public double avgMonth(int[] mlist){
    double sum = 0.0;
    for (int i=0; i < mlist.length; ++i)
        sum += mlist[i];
    return sum / mlist.length;
}
```

Two Dimensional Arrays

- Suppose we have six voters and we ask them a question answered as yes or no

<table>
<thead>
<tr>
<th>question</th>
<th>voter0</th>
<th>voter1</th>
<th>voter2</th>
<th>voter3</th>
<th>voter4</th>
<th>voter5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

represent as array of boolean

```java
boolean[] votes;
```

- Suppose we ask two more yes/no questions

<table>
<thead>
<tr>
<th>question0</th>
<th>voter0</th>
<th>voter1</th>
<th>voter2</th>
<th>voter3</th>
<th>voter4</th>
<th>voter5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>question1</th>
<th>voter0</th>
<th>voter1</th>
<th>voter2</th>
<th>voter3</th>
<th>voter4</th>
<th>voter5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>question2</th>
<th>voter0</th>
<th>voter1</th>
<th>voter2</th>
<th>voter3</th>
<th>voter4</th>
<th>voter5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

how to represent?

- It is a table with rows and columns
- an array of arrays
Java Two Dimensional Arrays

- A two dimensional array is just an array of arrays
  - That is, an array whose element type is an array
- Easiest to think of as table with rows and columns
  - Each row is an array
  - The table is an array of rows
  - The length of a row is the number of columns

- Java declaration syntax
  \[ boolean [ ] [ ] votes = new boolean [3][6]; \]

Access a value from a two dimensional array
- \( votes[0][0] \) is the first value in the first row
- \( votes[1][0] \) is the first value in the second row
- \( votes[0][5] \) is the last value in the first row
- \( votes[2][5] \) is the last value in the last row
- Use two indices to get an element of the 2D array

How does this work?
- Remember a 2D array is an array of arrays
- So if we use just a single index, we get an array
  - This is a row if we're picturing a table
  - First index is row index, second index is column index
- If we also use a second index, we get an element of the row
- This works because of the operator grouping: \( (votes[0])[0] \)
Multi-Dimensional Arrays

- All rows of a 2D array need not have the same length
  - A whole row could be replaced by:
    ```java
    votes[1] = new boolean[10];
    ```
  - Then middle row is twice as long as the others
- Since we can create an array of anything, we can have arrays of 2D arrays
  - I.e., three dimensional arrays (and arrays of 3D are 4D ...)
  - Use 3 dimensions for creation, and 3 indexes to access element
  - For example, track temperatures at points in rectangular space
    ```java
    double[][][] temps = new double[5][7][20];
    ```
    
    3D array
    
    length, width, height
    
    5 x 7 x 20 = 700 values
    
    ```java
    temps[2][5][13] = 15.5;
    ```
    
    Set value at a position

Command Line Arguments

- From our first program, we have seen the main method
  ```java
  public static void main(String [] args)
  ```
  - Now we can understand what it means and how to use it
- The parameter passed to main is an array of Strings
  - These strings are from ***command line arguments***
  - A Java program is executed by the java virtual machine
  - This may be done automatically (as in TextPad) or we may run
    the JVM and our program manually by typing a command in a
    Command Prompt window, or a terminal window in Unix
- The "words" on the command line after the class name
  are available to our program in the array of Strings
  - Words are separated by white space
  - If a "word" is a number, the program must convert it
  - The number of program arguments is the length of the array

- CommandLine.java
Copying Arrays

- Program logic may require an array to be copied
  - **Caution**: An array is an object. Using assignment only copies the reference, resulting in an alias for the same array.
  - Since an array is a mutable object, changes through one variable would be seen through an alias as well
- To make an actual copy of an array
  - Create a new array of the same type and the same size
  - Copy each element (loop over length of array)
  - If the elements are mutable objects (like arrays), they must be copied carefully (i.e., assignment is probably not sufficient)
  - We need to perform a deep copy of the array

Variable Length Parameter Lists

- Sometimes a method that we designed to work for two values would be useful for three, or four, or five ...
  - For example, a method that finds the maximum of two numbers, or a method that calculates the average of two numbers
- How do we generalize to more arguments?
  - Could define a new (overloaded) version of the method with 3 arguments, another with 4, etc.
    - This is very tedious, repetitive, error prone, and a maintenance problem
  - Could pass the method an array instead of individual arguments
    - But our calling code may not have the values in an array, and it could be a pain to create one just for the call
- **Solution**: Java allows **variable length parameter lists**
  - With this construct, Java creates an array for us automatically
  - Variable arguments must be of same type
Variable Length Parameter Lists

- **Definition syntax:**
  ```java
  double average(int ... numbers) {
    double result = 0.0;
    if (numbers.length == 0) return result;
    for (int i=0; i < numbers.length; ++i)
      result += numbers[i];
    return result / numbers.length;
  }
  ```

- **Calling the method:**
  ```java
  int x = 5, y = 7;
  double a = average(x, x*y, 13, 23, 3*y);
  ```

  [VariableArgs.java](VariableArgs.java)