Arrays

Introduction to Arrays

Processing Array Contents

Reassigning Array References

Two-Dimensional Arrays

The ArrayList Class

Introduction to Arrays

Introduction to Arrays

- An array is a **container** that holds multiple values instead of one.
- All values in an array must be of the same type.
- The number of values an array can hold is known as the **length**.
- Once the length of an array is set, it cannot be changed.
- We use integers to **index** into an array to get to the individual values.
- The first index position is always 0.

Introduction to Arrays

Arrays are built into Java much like objects. Here is the array syntax:

```
int[] numbers;
```

Like objects, you aren't declaring an array. You are declaring an array **reference**.

Introduction to Arrays

Like objects, arrays in Java are declared dynamically using the keyword **new**.

```
numbers = new int[6];
```

You may declare the array reference and the array in the same statements.

```
int[] numbers = new int[6];
```

The first index of an array is 0.

Initial Values

Unlike individual variables, the values in an array come with preset values which means you can use them without initialzing them. It's good to know what those values are.

- All elements in an numerical array (byte, short, int, long, float, and double) will be set to 0.
- All elements in an character array will be set to integer 0.
- All elements in a boolean array will be set to **false**.
- All elements in every other type of array will be set to **null**.

Arrays may be of any type.

```
float[] temperatures = new float[100];
char[] letters = new char[41];
long[] units = new long[50];
double[] sizes = new double[1200];
```

Creating arrays

- The array length must be a non-negative number.
- It may be a literal, a constant, or a variable.
- Once created, the array length is fixed and cannot be changed.

Code.

```
final int ARRAY_LENGTH = 6;
int[] numbers = new int[ARRAY_LENGTH];
```

Accessing the Elements of an Array

We can write to an array at an index position. The first index position is 0. The last index position is the length of the list minus 1.

```
numbers[0] = 42;
numbers[5] = 11;
```

- Before: [0,0,0,0,0,0]
- After: [42,0,0,0,0,11]
- Array elements can be treated as any other variable.
- You are not allowed to index into an array outside of the range of 0 to length-1. This will throw an error and your program will crash.

Bounds Checking

```
int[] values = new int[10];
```

- Array indexes always start at 0 and continue to the length of the list minus 1.
- What are the indicies for the above array?

Off-by-one errors

```
int[] numbers = new int[100];
for (int i = 1; i <= 100; i++) {
numbers[i] = i;
}
```

- An off-by-one error is when you happen to forget the bounds of the data that you are working with.
- What is wrong with the above code?
- This code will compile, but it will crash with an `ArrayIndexOutOfBoundsException`. Why?

Array Initialization

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

- When there are but a few items, you can initialize an array like this.
- days[0] is 31. days[1] is 28.
- There are 12 elements in this array.

Array Length

Java arrays will remember their own length.

```
int size = days.length; // size will be equal to 12
```

Let's write a programming exercise.

1. Ask the user for a number of elements.
2. Declare an array of that size.
3. Prompt the user for numbers to put into that array.
4. In a separate for-loop, compute the sum of that array.

Alternate Array Declaration

- Previously we showed arrays being declared:
  - int[] numbers;
- However, the brackets can also go here:
  - int numbers[];
  - These are equivalent but the first style is typical.

Alternate Array Declaration

- Multiple arrays can be declared on the same line.
  - int[] numbers, codes, scores;
- With the alternate notation each variable must have brackets.
  - int numbers[], codes[], scores;
  - The scores variable in this instance is simply an int variable.

Processing Array Contents

Processing Array Contents

Processing data in an array is the same as any other variable.

```
grossPay = hours[3] * payRate;
```

Pre and post increment works the same:

```
int[] score = {7, 8, 9, 10, 11};
++score[2]; // Pre-increment operation
score[4]++; // Post-increment operation
```

Processing Array Contents

Array elements can be used in relational operations:

```
if(cost[20] < cost[0]) { //statements }
```

They can be used as loop conditions:

```
while(value[count] != 0) { //statements }
```

Array Length

Arrays are objects and provide a public field named length that is a constant that can be tested.

```
double[] temperatures = new double[25];
```

The length of this array is 25. The length of an array can be obtained via its length constant.

```
int size = temperatures.length;
```

The variable size will contain 25.

The Enhanced for Loop

- Simplified array processing (read only)
- Always goes through all elements

Code.

```
for(datatype elementVariable : array)
statement;
```

The Enhanced for Loop

```
int[] numbers = {3, 6, 9};
for(int val : numbers) {
System.out.printf("The next value is %d.%n", val);
}
```

Array Size

The length constant can be used in a loop to provide automatic bounding.

```
for(int i = 0; i < temperatures.length; i++) {
  System.out.printf("Temperature %d: %d%n",
  i, temperatures[i]);
}
```

Reassigning Array References

Reassigning Array References

An array reference can be assigned to another array of the same type.

```
// Create an array referenced by the numbers variable.
int[] numbers = new int[10];

// Reassign numbers to a new array.
numbers = new int[5];
```

If the first (10 element) array no longer has a reference to it, it will be garbage collected.

Java Garbage Collection

- Java uses reference counting to keep track of the number of references which point to a block of memory.
- If the number of references on an object drops to zero, the **garbage collector thread** will delete it for you.
- You can set any reference in Java to **null** to reduce the references on an object to **one less**.
  - Usually "one less" means zero and the object will be automatically deleted.

Reassigning Array References

This is not the way to copy an array.

```
int[] array1 = { 2, 4, 6, 8, 10 };
int[] array2 = array1;
```

This does make two references to the array containing "2, 4, 6, 8, 10". There's only one array.

```
array1[0] = 15;
array2[1] = 20;
```

After these two lines execute, the array will now be "15, 20, 6, 8, 10".

Copying Arrays

- You cannot copy an array by merely assigning one reference variable to another.
- You need to copy the individual elements of one array to another.

Copy.

```
int[] one = {5, 10, 15, 20, 25 };
int[] two = new int[one.length];
for (int i = 0; i < one.length; i++)
    two[i] = one[i];
```

This code copies each element of **one** to the corresponding element of **two**. There are now two independent arrays.

Passing Array Elements to a Method

- When a single element of an array is passed to a method it is handled like any other variable.
- More often you will want to write methods to process array data by passing the entire array, not one element at a time.
- Arrays are objects.
- Their references can be passed to methods like any other object reference variable.

Comparing Arrays

- Like with String objects, the == operator determines only whether array references point to the same array object.
- Unfortunately this is not good for arrays and you must write your own code to directly compare objects (including array).

Comparing Arrays: Example

Let's turn this into a public static method. Here's the documentation.

```
/**
 * The method will return true if two arrays are identical
 *
 * Precondition: one and two must not be null
 * Postcondition: none
 *
 * @param one the first array to compare
 * @param two the second array to compare.
 * @return true if the arrays are equal, false otherwise
 */
```

Comparing Arrays: Example

```
public static boolean equals(int[] one, int[] two) {
    if (one == null) { return false; }
    if (two == null) { return false; }
    if (one.length != two.length) { return false; }
    boolean same = true;
    for (int i = 0; i < one.length; i++) {
        if (one[i] != two[i])
            same = false;
    }
    return same;
}
```

Useful Array Operations: Max

```
public static int maximum(int[] numbers) {
    int highest = numbers[0];
    for (int value : numbers) {
        if (value > highest)
            highest = value;
    }
    return highest;
}
```

Useful Array Operations: Min

```
public static int minimum(int[] numbers) {
    int lowest = numbers[0];
    for (int value : numbers) {
        if (value < lowest)
            lowest = value;
    }
    return lowest;
}
```

Useful Array Operations: Sum

```
public static int sum(int[] numbers) {
    int total = 0;
    for (int value : numbers) {
        total += value;
    }
    return total;
}
```

Useful Array Operations: Average

```
public static double average(int[] numbers) {
    return sum(numbers) / (double) numbers.length;
}
```

Notice here that we must cast the length of the numbers array to a double before diving.

Returning an Array Reference

- A method can return a reference to an array.
- The return type of the method must be declared as an array of the right type.

Code.

```
public static double[] getArray() {
  double[] array = { 1.2, 2.3, 4.5, 6.7, 8.9 };
  return array;
}
```

The getArray method is a public static method that returns an array of doubles.

String Arrays

- Arrays are not limited to primitive data.
- An array of String objects can be created:

Code.

```
String[] names = { "Bill", "Susan", "Steven", "Jean" };
```

String Arrays

If an initialization list is not provided, the new keyword must be used to create the array:

```
String[] names = new String[4];
```

- Every element in this array will be initialized to **null**.
- A single object reference will not even compile unless the reference is initialized.
- Unlike single object references, elements in a array are automatically initialized.

String Arrays

When an array is created in this manner, each element of the array must be initialized with a new String object.

```
for (int i = 0; i < names.length; i++) {
    names[i] = new String();
}
```

Calling String Methods On Array Elements

- String objects have several methods, including:
  - toUpperCase
  - compareTo
  - equals
  - charAt
- Each element of a String array is a String object.
- Methods can be used by using the array name and index as before.

Code.

```
System.out.println(names[0].toUpperCase());
char letter = names[3].charAt(0);
```

The length Field & The length Method

- Arrays have a final field named length.
- String objects have a method named length.
- To display the length of each string held in a String array:

Code.

```
for (int i = 0; i < names.length; i++)
  System.out.println(names[i].length());
```

- Notice the difference between the first "length" and the second "length()".
- An array's length is a field. You do not write a set of parentheses after its name.
- A String's length is a method. You do write the parentheses after the name of the String class's length method.

Arrays of Objects

Because Strings are objects, we know that arrays can contain objects.

```
BankAccount[] accounts = new BankAccount[5];
```

Arrays of Objects

Each element needs to be initialized.

```
for (int i = 0; i < accounts.length; i++)
    accounts[i] = new BankAccount();
```

The Sequential Search Algorithm

- A search algorithm is a method of locating a specific item in a larger collection of data.
- The sequential search algorithm uses a loop to:
  - sequentially step through an array,
  - compare each element with the search value, and
  - stop when the
    - value is found or
    - the end of the array is encountered.

Sequential Search Algorithm

```
public static boolean search(int[] array, int toFind) {
    if (array == null) {
        throw new IllegalArgumentException
            ("array is null");
    }
    boolean found = false;
    for (int value : array) {
        if (value == toFind) {
            found = true;
        }
    }
    return found;
}
```

Two-Dimensional Arrays

Two-Dimensional Arrays

- A two-dimensional array is an array of arrays.
- It can be thought of as having rows and columns.
- Declaring a two-dimensional array requires two sets of brackets and two size declarators
  - The first one is for the number of rows
  - The second one is for the number of columns.

Code.

```
double[][] scores = new double[3][4];
```

- The two sets of brackets in the data type indicate that the scores variable will reference a two-dimensional array.
- Notice that each size declarator is enclosed in its own set of brackets. Two-Dimensional Arrays

Accessing Two-Dimensional Array Elements

- When processing the data in a two-dimensional array, each element has two subscripts:
  - one for its row and
  - another for its column.

Accessing Two-Dimensional Array Elements

- Programs that process two-dimensional arrays can do so with nested loops.

Code.

```
for (int row = 0; row < 3; row++) {
  for (int col = 0; col < 4; col++) {
    System.out.print("Enter a score: ");
    scores[row][col] = keyboard.nextDouble();
  }
}
```

Accessing Two-Dimensional Array Elements

To print out the scores array:

```
for (int row = 0; row < 3; row++) {
  for (int col = 0; col < 4; col++) {
    System.out.println(scores[row][col]);
  }
}
```

Initializing a Two-Dimensional Array

- Initializing a two-dimensional array requires enclosing each row's initialization list in its own set of braces.
  - int[][] numbers = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9} };
- Java automatically creates the array and fills its elements with the initialization values.
  - row 0 {1, 2, 3}
  - row 1 {4, 5, 6}
  - row 2 {7, 8, 9}
- Declares an array with three rows and three columns.

The length Field

- Two-dimensional arrays are arrays of one-dimensional arrays.
- The length field of the array gives the number of rows in the array.
- Each row has a length constant tells how many columns is in that row.
- Each row can have a different number of columns.

Displaying The Elements of a Two-Dimensional Array

```
public static void array2dShow(int[][] array) {
    for (int[] row : array){
        for (int cell : row){
            System.out.printf(" %3d", cell);
        }
        System.out.println();
    }
}
```

Summing The Elements of a Two-Dimensional Array

```
public static int array2dSum(int[][] array) {
    int total;
    for (int[] row : array){
        for (int cell : row){
            total += cell;
        }
    }
    return total;
}
```

Summing The Row of a Two-Dimensional Array

```
public static int[] array2dRow(int[][] array) {
    int[] total = new int[array.length];
    for (int row = 0; row < array.length; row++){
        for (int cell : array[row]){
            total[row] += cell;
        }
    }
    return total;
}
```

Summing The Columns of a Two-Dimensional Array

```
public static int[] array2dCol(int[][] array) {
    int[] total = new int[array[0].length];
    for (int col = 0; col < array[0].length; col++){
        for (int row = 0; row < array.length; row++)
            total[col] += array[row][col];
    }
    return total;
}
```

Passing and Returning Two-Dimensional Array References

- There is no difference between passing a single or two-dimensional array as an argument to a method.
- The method must accept a two-dimensional array as a parameter.

Ragged Arrays

- When the rows of a two-dimensional array are of different lengths, the array is known as a ragged array.
- You can create a ragged array by creating a two-dimensional array with a specific number of rows, but no columns.
  - int[][] ragged = new int [4][];
- Then create the individual rows.

Code.

```
ragged[0] = new int[3];
ragged[1] = new int[4];
ragged[2] = new int[5];
ragged[3] = new int[6];
```

More Than Two Dimensions

- Java does not limit the number of dimensions that an array may be.
- More than three dimensions is hard to visualize, but can be useful in some programming problems.

Command-Line Arguments

- A Java program can receive arguments from the operating system command-line.
- The main method has a header that looks like this:
  - public static void main(String[] args)
- The main method receives a String array as a parameter.
- The array that is passed into the args parameter comes from the operating system command-line.
- It is not required that the name of main's parameter array be **args**.
  - By convention, it almost always is.

Variable-Length Argument Lists

- Special type parameter - type...
- Vararg parameters are arrays

Code.

```
public static int sum(int... numbers) {
    int total = 0; // Accumulator
    // Add all the values in the numbers array.
    for (int val : numbers)
        total += val;
    // Return the total.
    return total;
}
```

Variable-Length Argument Lists

When we call the method in the last slide, we can use as many integers as we want.

```
System.out.println(sum(1,2,3,4)); // Prints 10
System.out.println(sum(1,2,3));   // Prints 6
System.out.println(sum(1,2));     // Prints 3
System.out.println(sum(1));       // Prints 1
```

The ArrayList Class

The ArrayList Class

- Similar to an array, an ArrayList allows object storage
- Unlike an array, an ArrayList object:
  - Automatically expands when a new item is added
  - Automatically shrinks when items are removed
- Requires: java.util.ArrayList import statement. (Get Netbeans to fix this for you.)

Creating an ArrayList

- To populate the ArrayList, use the add method:
  - nameList.add(“James”);
  - nameList.add(“Catherine”);
- To get the current size, call the size method
  - nameList.size(); // returns 2

Using an ArrayList

To access items in an ArrayList, use the get method

```
nameList.get(1);
```

In this statement 1 is the index of the item to get.

Using an ArrayList

The ArrayList class's toString method returns a string representing all items in the ArrayList

```
System.out.println(nameList);
```

This statement yields: [James, Catherine]

Using an ArrayList

The ArrayList class's remove method removes designated item from the ArrayList

```
nameList.remove(1);
```

This statement removes the second item.

Using an ArrayList

- The ArrayList class's add method with one argument adds new items to the end of the ArrayList
- To insert items at a location of choice, use the add method with two arguments:

Using an ArrayList

This statement inserts the String “Mary” at index 1.

```
nameList.add(1, "Mary");
```

This statement replaces “Mary” with “Becky”.

```
nameList.set(1, "Becky");
```

Using an ArrayList

- An ArrayList has a capacity, which is the number of items it can hold without increasing its size.
- The default capacity of an ArrayList is 10 items.
- To designate a different capacity, use a parameterized constructor:

Code.

```
ArrayList<String> list = new ArrayList<String>(100);
```

Using an ArrayList

- You can store any type of object in an ArrayList.