Lecture 27:
1-D Arrays, Part I

Sierra College
CSCI-12
Spring 2015
Weds 05/06/15

Announcements

- **General**
  - Added lab hours, tentatively:
    - **Friday 5/15   9am-noon** (for finish-up work on Dam class, or LAST program)
    - **Friday 5/22   10am-2pm** (for work on LAST program)
- **Schedule**
  - 3 more lectures, then finals week (final exam Weds 5/20)
    - Review session Monday, study guide, in-class exam (like last time)
  - Current program, then one LAST one (due after final)
- **Past due assignments**
  - PRGM22: Menu For Demo (accepted thru **tonight** @ 11pm)
- **Current assignments**
  - PRGM25: Dam (due Thurs 5/14 @ 11pm)
    - Create a new class which models a water storage dam
    - Use the systematic procedure we have gone thru in lectures
    - Refer back to prior lecture notes

Lecture Topics

- **Last time**:
  - Finished up our ***Person*** class
    - Data safety
    - Safe transfer of object data
    - Utility methods
    - Testing considerations
- **Today**
  - New topic: 1-D arrays
    - Definitions
    - Creating and initializing
    - Array mechanics
    - Common array operations

For Next Time

- **Lecture Prep**
  - Text readings and lecture notes
- **Program**
  - Continue building up your ***Dam*** class
    - Review the Writing Classes lecture notes
    - Review the assignment and the ***Dam*** API carefully (questions??)
    - Start with an empty Java class, and create the class pseudocode
    - Build a little, test a little (add test code to *main*() as you go)
    - Implement the entire starter class for ONE instance variable

Motivation for Arrays

- Suppose we had to archive and process a large amount of data
- One approach, of course, would be to treat each piece of data as an individual variable value (a “scalar”)
- But such an approach quickly becomes “painful”
  - We’d have 365 individual ***int*** variables lying around
- Is there a better way??

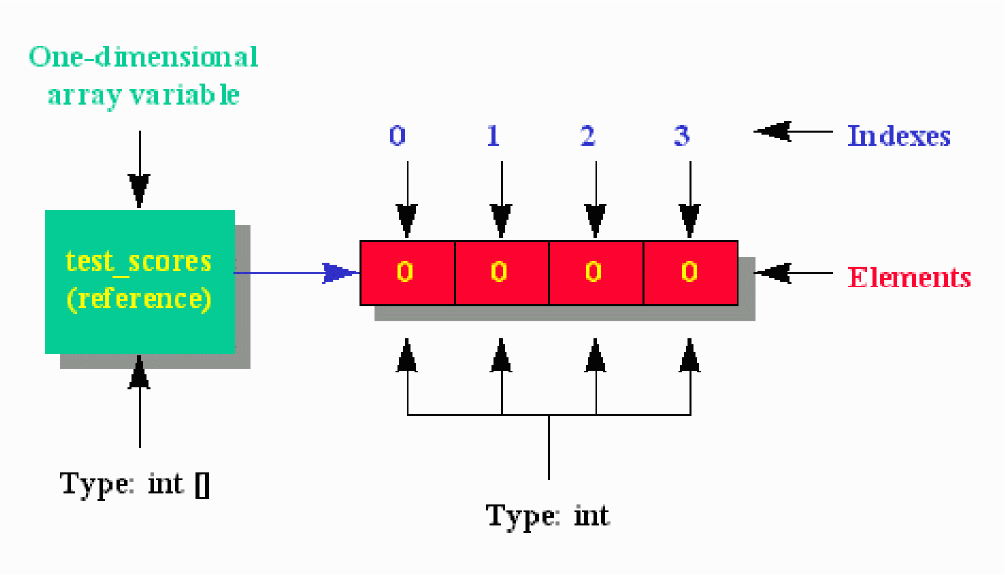
```
// temperature data in Sac for one year
double tempJan01 = 42.1;
double tempJan02 = 43.6;
….
double tempJul18 = 103.5;
double tempJul19 = 104.2;
….
double tempDec30 = 37.8;
double tempDec31 = 39.4;

// average temperature
double sum = tempJan01 + tempJan02 + … +
tempDec30 + tempDec31;
double avgYearlyTemp = sum / 365;
```

What Are Arrays?

- An **array** is a named, ordered sequence of variables/objects of the same datatype
- **Array elements** may be either:
  - Any of the 8 primitive datatypes
  - Objects of any class type (*String*, *SimpleDate*, *Person*, ***Dam***, …)
  - Other arrays themselves (this leads us to 2-D, 3-D, … arrays)
- Arrays allow us to refer to the collection of related variables…
  - In an aggregate way
  - With one common name
  - Using numerical indexing
- Manipulation and traversal of arrays is done thru the use of ***for*** loops
  - An array "knows" its own size, so a count-controlled loop is ideal

Some Array Terminology

- Each variable in an array is called an **element**
- Each element in an array may be accessed by its (0-based) **index**
- We declare and use an array using an **array reference**



Why Use Arrays?

- **Arrays** are a common programming element in countless applications
- Often useful to perform the same operations on long lists of like-typed items
- Commonplace in bulk data processing, mathematics, graphing, visualization, simulation, signal processing, image processing, statistics, etc…
- Arrays save us from having to create and maintain multiple scalar variables for data quantities
  - **Painful**: int score1, score2, score3, … , score99, score100
  - **Easy**: score[i] for i=0 → 99 (use a *for* loop)

Array Mechanics

- There is a concept elsewhere in programming called **"CRUD": Create, Read, Update, Delete**
- We will borrow this idea to outline the basic operations necessary for arrays
- **Create**: declaring, instantiating, and initializing arrays
- **Read**: accessing individual elements, looping thru or printing an entire array
- **Update**: modifying individual elements, or all elements in a loop
- Delete: (N/A for our discussion)

- For all the following slides, see the posted example file: ***Arrays1DExamples.java*** in **Example Source Code**

Creating Arrays

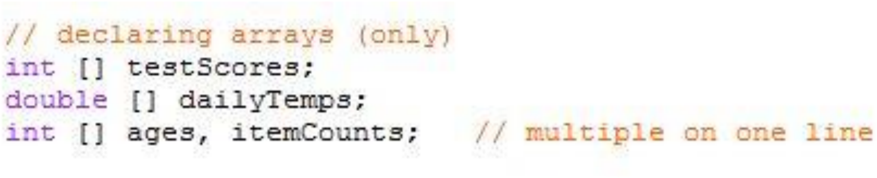
- Arrays are ultimately implemented as **objects**
- There are 3 steps to creating a “workable” array:
  - **Declaring** the array
    - States that an array “is going to exist”
    - Results in an array reference
  - **Instantiating** the array
    - States how large the array will be
    - Results in memory allocation for the array
    - Default values are given to the array elements
  - **Initializing** the array
    - Assigns specified values to array elements
    - Results in an array with desired (non-default) initial values

Declaring Arrays

- General form:

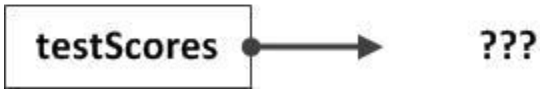
  datatype [ ] arrayName;

- Interpretation:
  - The “[]” is “part of the datatype”
  - “An *int* array”
  - “A *double* array”



```
// declaring arrays (only)
int [] testScores;
double [] dailyTemps;
int [] ages, itemCounts;   // multiple on one line
```

- Results in an **array reference**
  - It only “points to” an array
  - Not the actual array itself (yet)
  - Don’t even know yet how big the array will be
  - Similar to an object reference

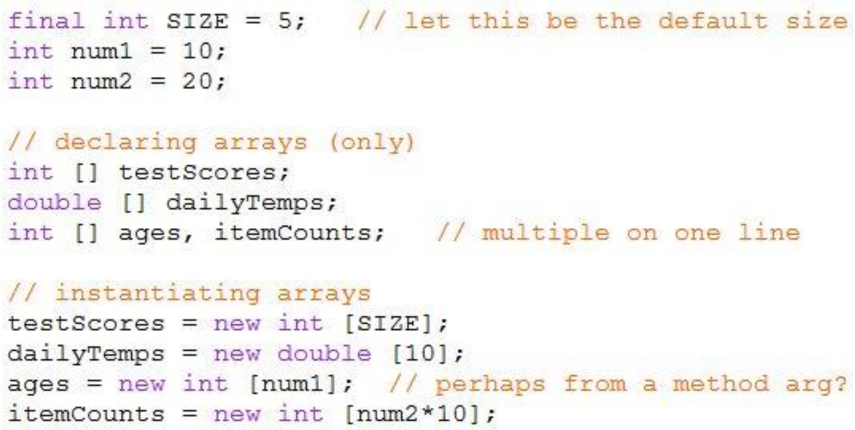


Instantiating Arrays

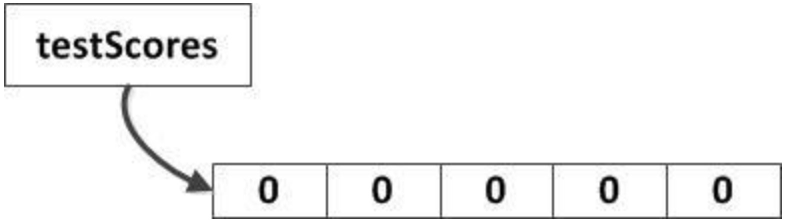
- General form:

  arrayName = new datatype [size]

- Memory allocation is performed (uses *new* keyword)
  - Memory size can be any integer expression
  - The memory size can be dynamically established, i.e.:
    - From a method argument
    - Calculated at runtime
    - In a class constructor (next program…)
- **Default values** are assigned to all array elements
  - We may wish to override these with our own specific values



```
final int SIZE = 5;   // let this be the default size
int num1 = 10;
int num2 = 20;

// declaring arrays (only)
int [] testScores;
double [] dailyTemps;
int [] ages, itemCounts;   // multiple on one line

// instantiating arrays
testScores = new int [SIZE];
dailyTemps = new double [10];
ages = new int [num1];   // perhaps from a method arg?
itemCounts = new int [num2*10];
```



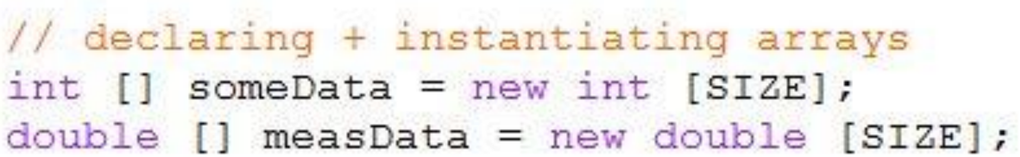
Default Values for Array Elements

Array data type	**Default value**
byte, short, int, long	0
float, double	0.0
char	The *null* character
boolean	*false*
Any object reference (for example, a *String*)	*null*

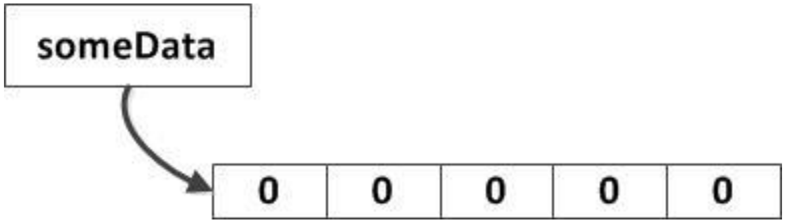
These defaults are exactly the same as for class instance variable defaults

Declaring AND Instantiating Arrays

- Arrays can be both declared AND instantiated in the same operation
  - This is perhaps more typical
- Just remember that there are two separate and distinct operations involved

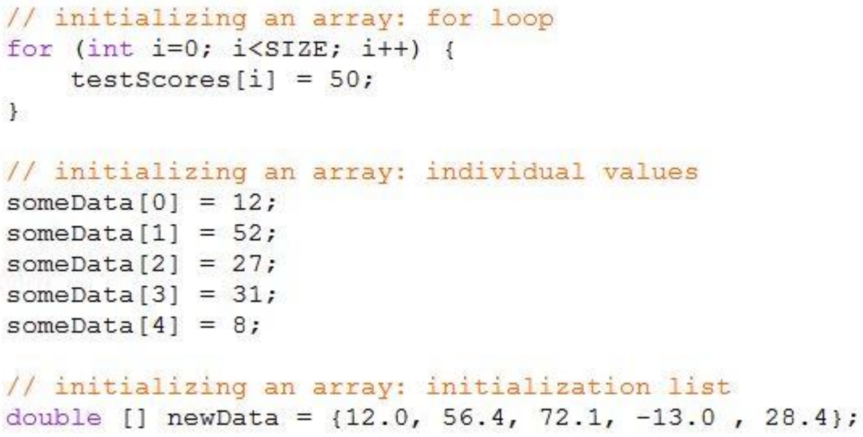


```
// declaring + instantiating arrays
int [] someData = new int [SIZE];
double [] measData = new double [SIZE];
```

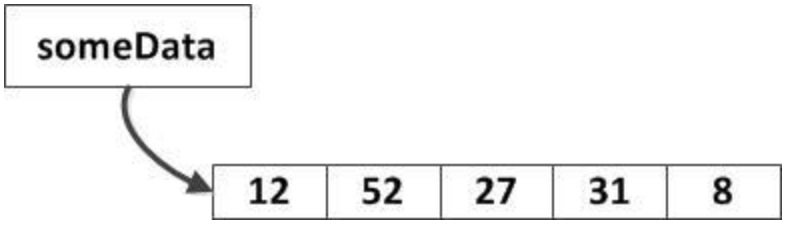


Initializing Arrays

- There are 3 ways of **initializing an array:**
- Using a ***for* loop**
  - Set all values to a constant, or some algorithmic value
- Setting **individual values**
  - Each element is index-accessed
  - Useful if random values, or if only certain values need non-default values
- Using an **initialization list**
  - Can only be done when array is declared
  - No *new* keyword or explicit size is used
  - Array is auto-sized according to the # of data elements provided



```
// initializing an array: for loop
for (int i=0; i<SIZE; i++) {
    testScores[i] = 50;
}

// initializing an array: individual values
someData[0] = 12;
someData[1] = 52;
someData[2] = 27;
someData[3] = 31;
someData[4] = 8;

// initializing an array: initialization list
double [] newData = {12.0, 56.4, 72.1, -13.0 , 28.4};
```



Array Elements and Boundaries

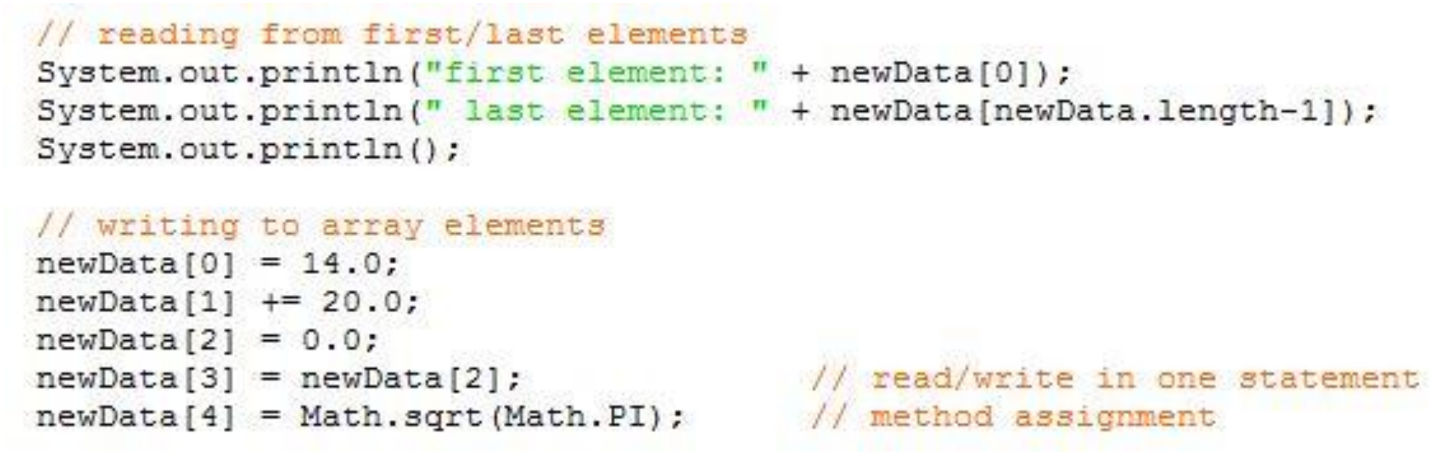
- Individual array elements are accessed via their numeric index within the array
  - Size of an array: **arrayName.length**
    - length is a read-only, integer instance variable of the array
    - Like *Strings*, arrays are 0-based
    - Different from *Strings*, which have: length strName.**length()**
  - First array element: **arrayName[0]**
  - General array element: **arrayName[i]**
  - Last array element: **arrayName[arrayName.length-1]**
- Attempting to index into an array beyond the above bounds results in a runtime error:

  ArrayIndexOutOfBoundsException

Reading/Writing Array Elements

- The individual elements of an array are each nothing more than a scalar variable of that particular datatype

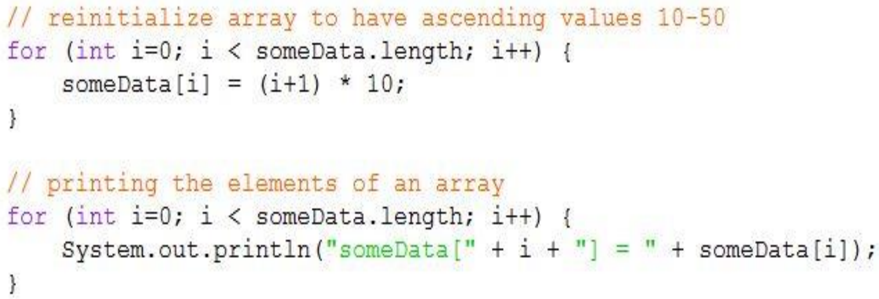
  arrayName[i] ← **entire expression is equivalent to one variable**
- They may be written to, read from, or otherwise manipulated just as any other variables of that datatype
- We **reference** them via their **index** within the array



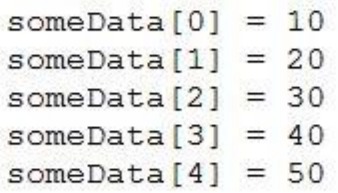
```
// reading from first/last elements
System.out.println("first element: " + newData[0]);
System.out.println(" last element: " + newData[newData.length-1]);
System.out.println();

// writing to array elements
newData[0] = 14.0;
newData[1] += 20.0;
newData[2] = 0.0;
newData[3] = newData[2];                // read/write in one statement
newData[4] = Math.sqrt(Math.PI);        // method assignment
```

Traversing An Array

- **Traversing an array** is done via a *for* loop
  - First index is 0
  - Last index is arrayName.length-1
  - **So loop up to: < arrayName.length**
- Arrays and *for* loops are inherently tightly coupled
- The loop index has **local scope** within the loop body, and can be used in the calculation of array values
- This would also be the means of **printing each element of an array**



```
// reinitialize array to have ascending values 10-50
for (int i=0; i < someData.length; i++) {
    someData[i] = (i+1) * 10;
}

// printing the elements of an array
for (int i=0; i < someData.length; i++) {
    System.out.println("someData[" + i + "] = " + someData[i]);
}
```



```
someData[0] = 10
someData[1] = 20
someData[2] = 30
someData[3] = 40
someData[4] = 50
```

Common Array Operations

- **Arrays** are ideal when we want to perform the same operations on each **element** of a collection of like-datatyped data
- Unfortunately, Java does not yet allow us to operate upon an entire array at once
- Thus, we need to use ***for*** loops to iterate through, and then operate upon, each **element** of the array in succession
- The following slides will demonstrate various standard, “cookbook” patterns with arrays

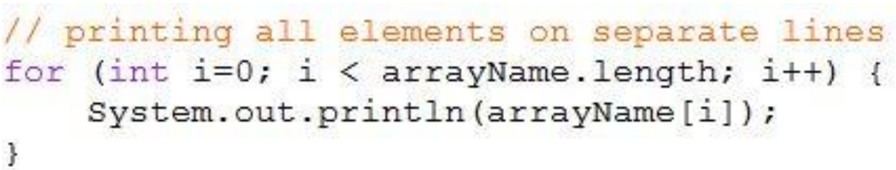
- See ***Arrays1DExamples.java*** in **Example Source Code**

Common Array Operations

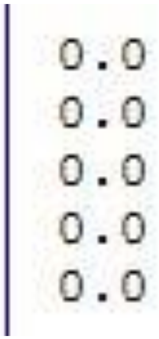
- Here are some standard, “typical” array operations:
  - Printing: printing all values
  - Initializing: setting all values (constant or algorithmic)
  - Reading: importing data from the cmd line, or from a file
  - Summing: total of all values
  - Average: combines summation and counting
  - Min/Max identifying min/max, and where it occurs
  - Copying copying an array’s contents to another array
  - Resizing increasing the size of an array
  - Equality do two arrays have same the contents?
  - Counters counters on an ordered group of outcomes

Printing An Array

Printing on separate lines

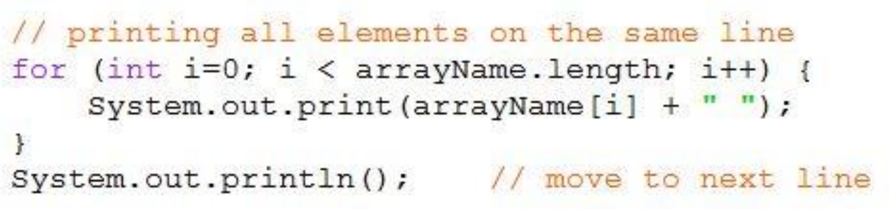


```
// printing all elements on separate lines
for (int i=0; i < arrayName.length; i++) {
    System.out.println(arrayName[i]);
}
```

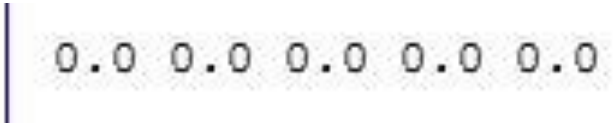


```
0.0
0.0
0.0
0.0
0.0
```

Printing on the same line



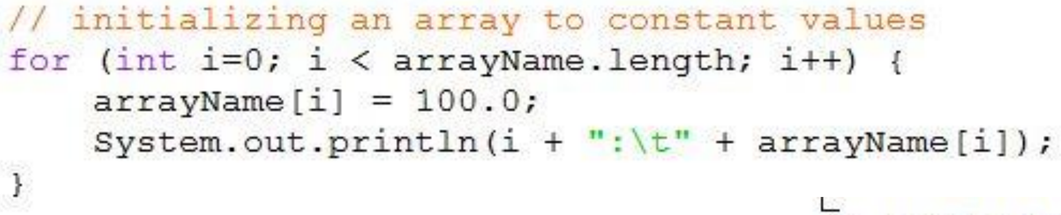
```
// printing all elements on the same line
for (int i=0; i < arrayName.length; i++) {
    System.out.print(arrayName[i] + " ");
}
System.out.println();    // move to next line
```



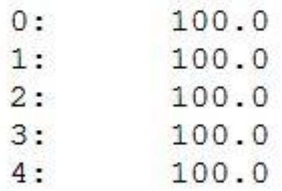
```
0.0 0.0 0.0 0.0 0.0
```

Initializing An Array

Initializing to a constant value

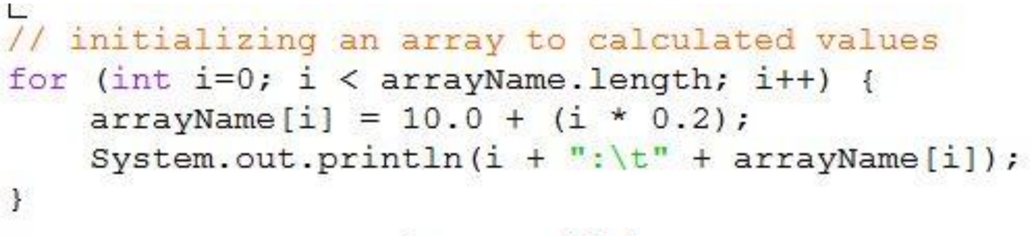


```
// initializing an array to constant values
for (int i=0; i < arrayName.length; i++) {
    arrayName[i] = 100.0;
    System.out.println(i + ":\t" + arrayName[i]);
}
```

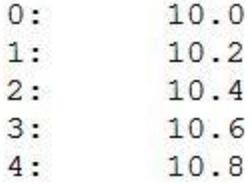


```
0:      100.0
1:      100.0
2:      100.0
3:      100.0
4:      100.0
```

Initializing to a calculated value

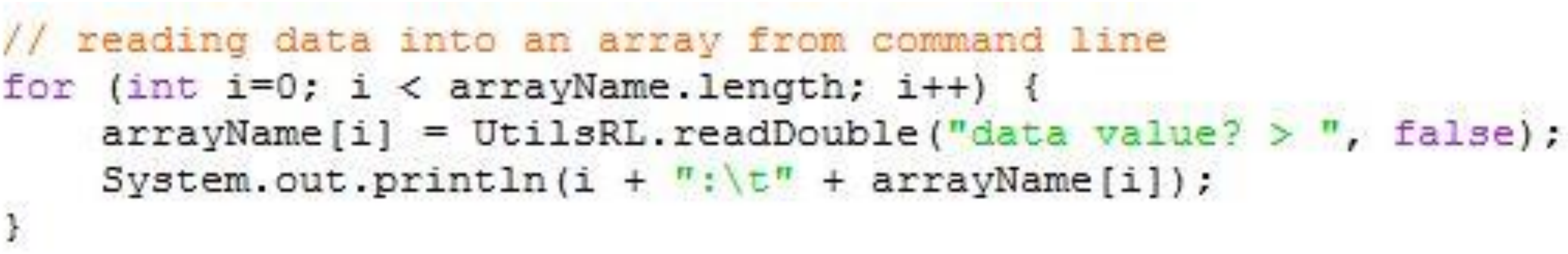


```
// initializing an array to calculated values
for (int i=0; i < arrayName.length; i++) {
    arrayName[i] = 10.0 + (i * 0.2);
    System.out.println(i + ":\t" + arrayName[i]);
}
```

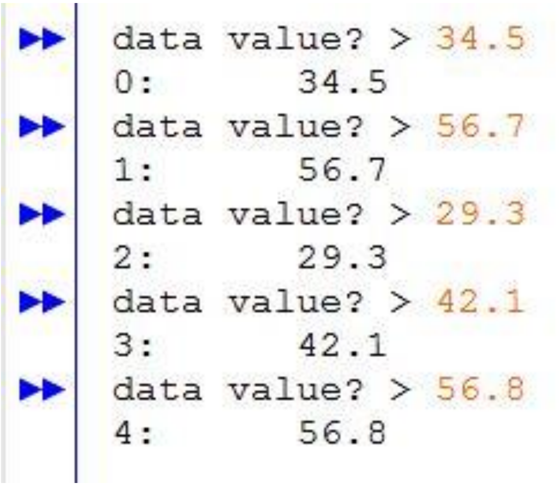


```
0:      10.0
1:      10.2
2:      10.4
3:      10.6
4:      10.8
```

Reading Data Into An Array

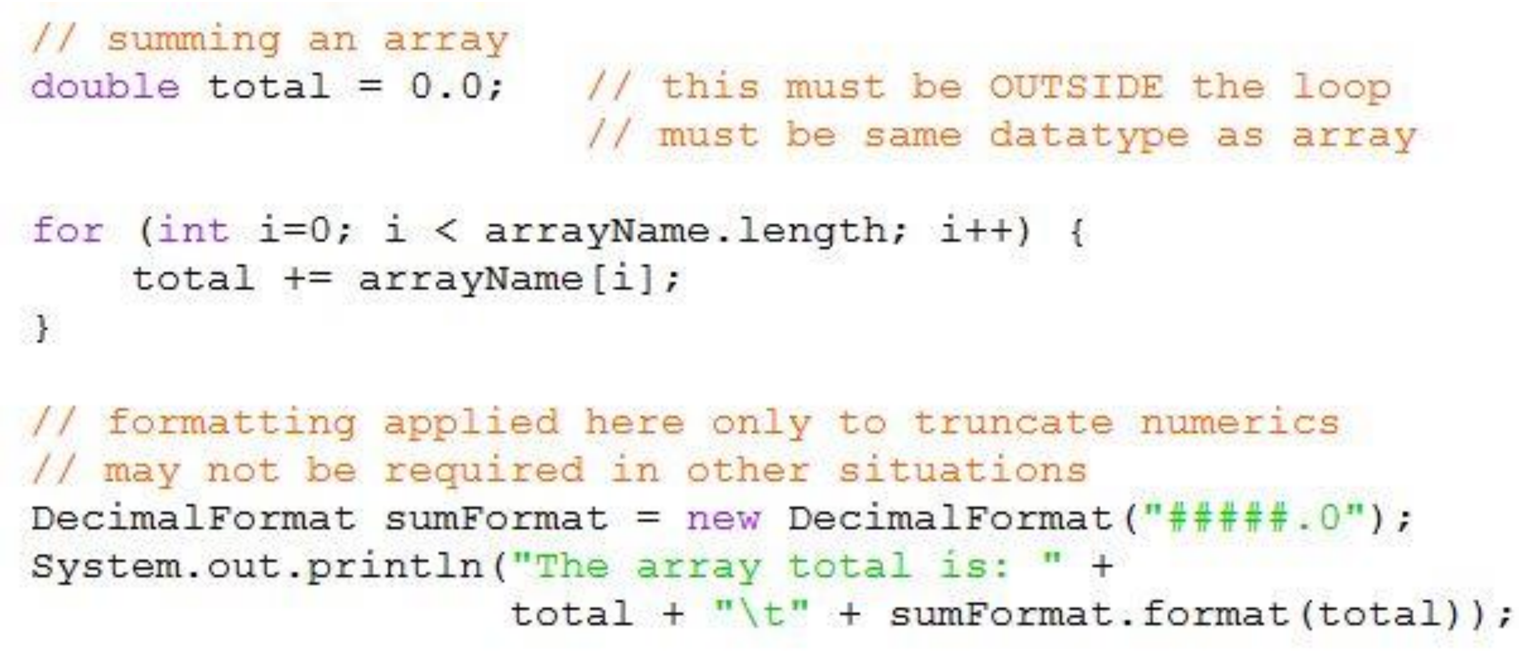


```java
// reading data into an array from command line
for (int i=0; i < arrayName.length; i++) {
    arrayName[i] = UtilsRL.readDouble("data value? > ", false);
    System.out.println(i + ":\t" + arrayName[i]);
}
```

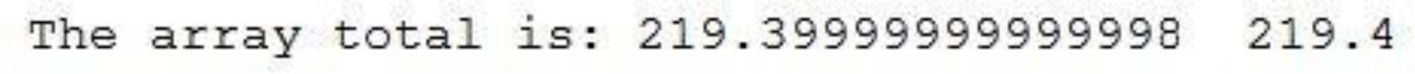


```
▶▶  data value? > 34.5
    0:      34.5
▶▶  data value? > 56.7
    1:      56.7
▶▶  data value? > 29.3
    2:      29.3
▶▶  data value? > 42.1
    3:      42.1
▶▶  data value? > 56.8
    4:      56.8
```

Summing An Array

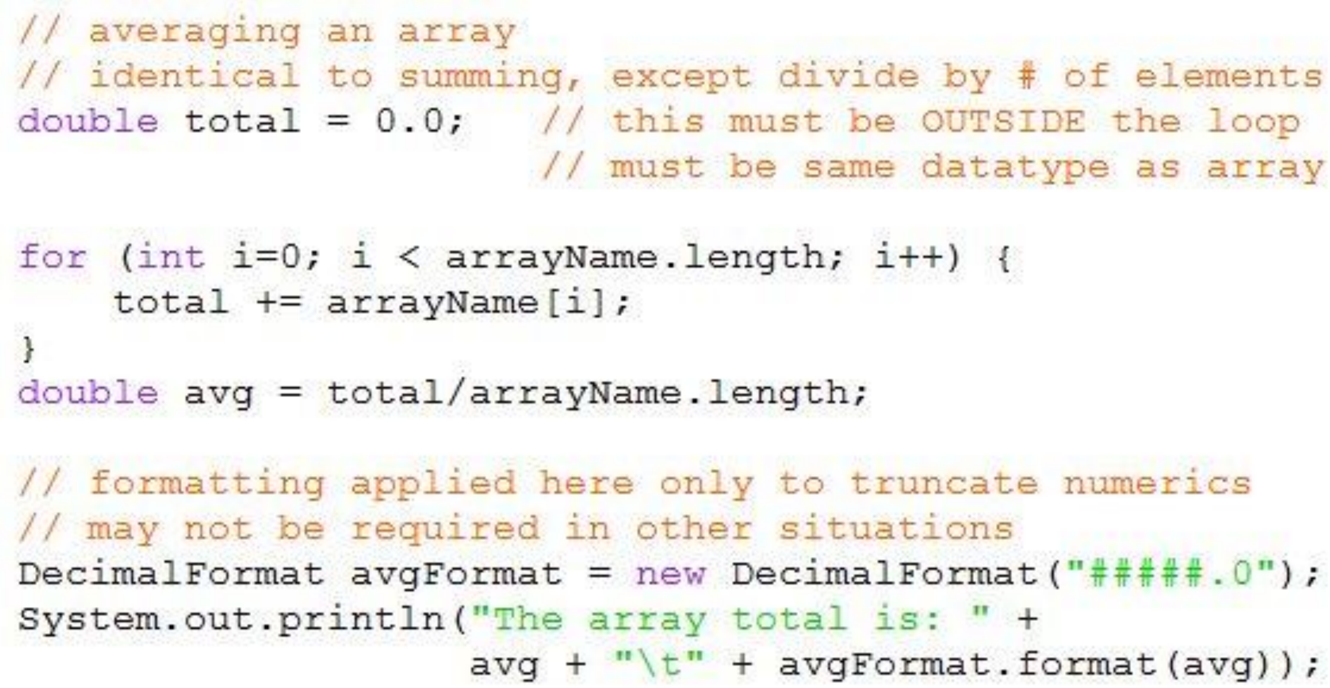


```
// summing an array
double total = 0.0;    // this must be OUTSIDE the loop
                       // must be same datatype as array

for (int i=0; i < arrayName.length; i++) {
    total += arrayName[i];
}

// formatting applied here only to truncate numerics
// may not be required in other situations
DecimalFormat sumFormat = new DecimalFormat("#####.0");
System.out.println("The array total is: " +
                   total + "\t" + sumFormat.format(total));
```

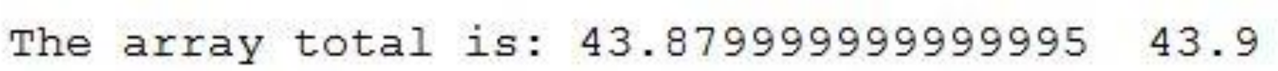


```
The array total is: 219.39999999999998  219.4
```

Averaging An Array

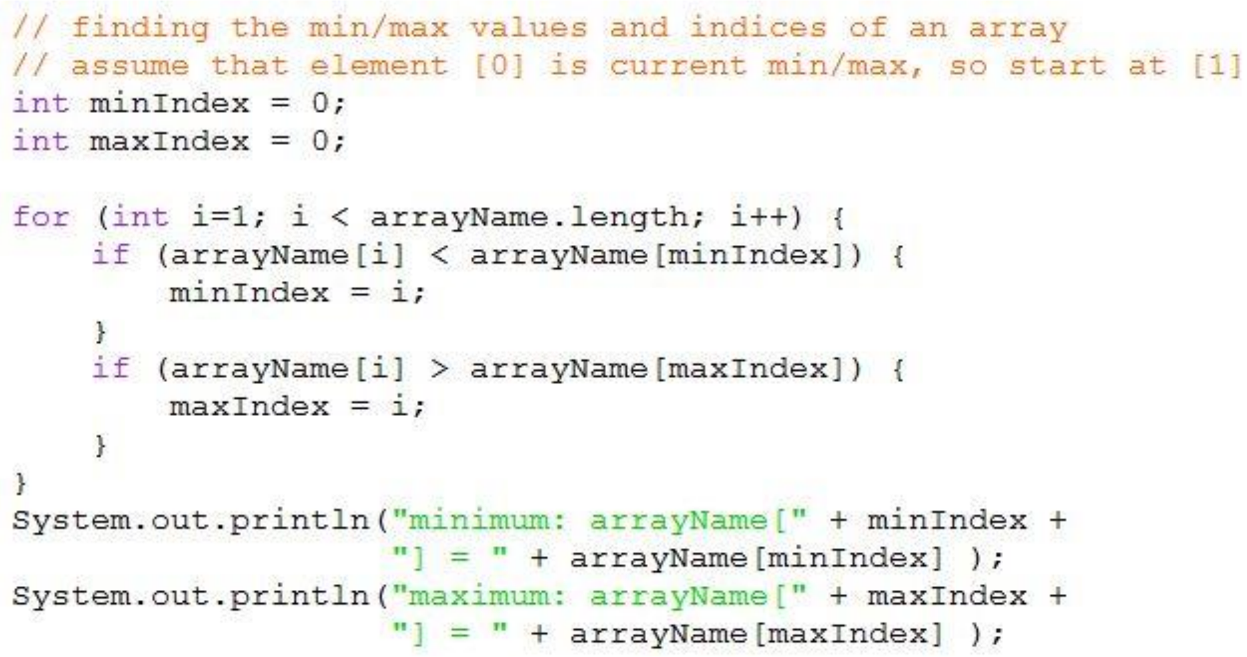


```
// averaging an array
// identical to summing, except divide by # of elements
double total = 0.0;    // this must be OUTSIDE the loop
                       // must be same datatype as array

for (int i=0; i < arrayName.length; i++) {
    total += arrayName[i];
}
double avg = total/arrayName.length;

// formatting applied here only to truncate numerics
// may not be required in other situations
DecimalFormat avgFormat = new DecimalFormat("#####.0");
System.out.println("The array total is: " +
                   avg + "\t" + avgFormat.format(avg));
```

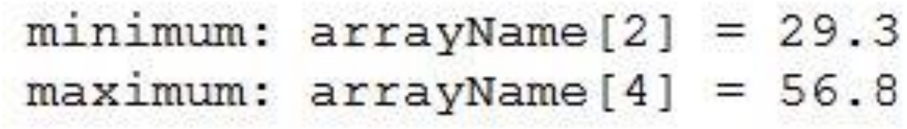


```
The array total is: 43.879999999999995  43.9
```

Finding Min/Max Of An Array



```
// finding the min/max values and indices of an array
// assume that element [0] is current min/max, so start at [1]
int minIndex = 0;
int maxIndex = 0;

for (int i=1; i < arrayName.length; i++) {
    if (arrayName[i] < arrayName[minIndex]) {
        minIndex = i;
    }
    if (arrayName[i] > arrayName[maxIndex]) {
        maxIndex = i;
    }
}
System.out.println("minimum: arrayName[" + minIndex +
                   "] = " + arrayName[minIndex] );
System.out.println("maximum: arrayName[" + maxIndex +
                   "] = " + arrayName[maxIndex] );
```



```
minimum: arrayName[2] = 29.3
maximum: arrayName[4] = 56.8
```