Arrays are the fundamental mechanism in Java for collecting multiple values.

An array collects a sequence of values of the same type.

double[] values = new double[10]; 10 is called the length of the array.

When you declare an array, you can specify the initial values. For example,

double[] moreValues = { 32, 54, 67.5, 29, 35, 80, 115, 44.5, 100, 65 }; When you supply initial values, you don't use the new operator

Individual elements in an array are accessed by an integer index i, using the notation array[i]. i = length -1

An array element can be used like any variable.

Attempting to access an element whose index is not within the valid index range is called a **bounds error**. The compiler does not catch this type of error. When a bounds error occurs at run time, it causes a run-time exception.

A bounds error, which occurs if you supply an invalid array index, can cause your program to terminate.

values.length yields the length of the values array.

Arrays suffer from a significant limitation: their length is *fixed*

int[] values = scores; // Copying array reference, scores is the original array

Because they share the same data that is stored in memory, modifying the array with either variable will affect another

only a part of the array will be occupied by actual elements. We call such an array **a partially filled array**.

With a partially filled array, keep a companion variable for the current size.

You can use the *enhanced* for loop to visit all elements of an array. enhanced for loop if you do not need the index values in the loop body.

for (double element : values) { total = total + element; }  
Common Array Algorithms:

**Filling** : values[i] = i \* I;

**Sum and Average**: total = total + element; total / values.length;

**Maximum and Minimum :** double largest = values[0]; if (values[i] > largest)

{ largest = values[i]; }

**Element Separators:** if (i > 0) { System.out.print(“ | ”); } System.out.print(values[i]);

or: System.out.print (Arrays.toString(values) );

**Linear Search**

while (pos < values.length && !found)

if (values[pos] == searchedValue) { found = true; } else { pos++; }

**Removing an Element**

for (int i = pos + 1; i < currentSize; i++) { values[i - 1] = values[i];} currentSize-- ;

**Inserting an Element**

if (currentSize < values.length){ currentSize++; values[currentSize - 1] = newElement;}

Before inserting an element, move elements to the end of the array starting with the last one.

if (currentSize < values.length){ currentSize++; for (int i = currentSize - 1; i > pos; i--) { values[i] = values[i - 1]; } values[pos] = newElement;}

**Swapping Elements**

Use a temporary variable when swapping two elements.

double temp = values[i]; values[i] = values[j]; values[j] = temp;

**Copying Arrays**

Use the *Arrays.copyOf* method to copy the elements of an array into a new array:

double[] prices = Arrays.copyOf(values, values.length);

Arrays.copyOf(values, n) allocates an array of length n, copies the first n elements of values

The following is using a bigger array to take the place of the old array.

double[] newValues = Arrays.copyOf(values, 2 \* values.length);

double[] newValues = Arrays.copyOf(values, n)

double[] newValues = new double[n];

for (int i = 0; i < n && i < values.length; i++){ newValues[i] = values[i];}

**Reading Input**

add the inputs to an array until the end of the input has been reached.

int currentSize = 0; while (in.hasNextDouble() && currentSize < inputs.length)

{inputs[currentSize] = in.nextDouble(); currentSize++;} inputs = Arrays.copyOf(inputs, currentSize);

**Sorting with the Java Library**

Arrays.sort(values, 0, currentSize);

Arrays can occur as method arguments and return values.

public static double [] sum(double[] values)<- expects an array and return an array

**ArrayList** class supplies methods for common tasks, such as inserting and removing elements. can grow and shrink as needed

ArrayList<String> names = new ArrayList<String>();

Use the size method to obtain the current size of an array list.

names.add();

Use the *get* methods to access an array list element at a given index.

names.get(2);

To set an array list element to a new value, use the set method. The set method overwrites existing values names.set(2, “Carolyn”);

Enhanced for Loop with Array Lists : for (String name : names)

ArrayList<String> friends = names; copy by reference

ArrayList<String> newNames = new ArrayList<String>(names); full copy of data

Input and output array list in method:

public static ArrayList<String> reverse(ArrayList<String> names)