APCS Arrays

I. Array basics

- **Java stores lists of values in arrays. An array is contiguous group of related memory locations. These locations are related by the fact that they all have the same name and the same type. To refer to a particular location or element within the array, we specify the name of the array and the index, or position, number of the element.**

- **Java arrays always begin with element 0; thus it is important to note the difference when referring to the "seventh element of the array" as opposed to "array element seven". The seventh element has a subscript of 6, while array element seven has a subscript of 7 (actually the eighth element of the array).**

- **Arrays occupy space in memory and are considered to be objects. Operator `new` must be used to reserve memory space for an array. For example, the following statements create an array of 100 `double` values called `temperature`:**

```
double[] temperature = new double[100];
```

or

```
double temperature[] = new double[100];
```

Look at the following simple program:

```
public class Array
{
    public static void main( String[] args )
    {
        int b[] = new int[5];
        for(int k=0; k<b.length; k++) b[k] = k+1;
        for(int j=0; j<b.length; j++) System.out.print("b[" + j + "] = " + b[j] + "\t");
        System.out.println("\n");
        int[] c = new int[5];
        for(int k=0; k<c.length; k++) c[k] = k+2;
        for(int j=0; j<c.length; j++) System.out.print("c[" + j + "] = " + c[j] + "\t");
        int sum = 0;
        for(int index=0; index<c.length; index++) sum += c[index];
        System.out.println("\n\nThe sum of all the elements of array \"c\" is " +
                           sum + ".");
    }
}
```

OUTPUT:

```
b[0] = 1    b[1] = 2    b[2] = 3    b[3] = 4    b[4] = 5

c[0] = 2    c[1] = 3    c[2] = 4    c[3] = 5    c[4] = 6

The sum of all the elements of array "c" is 20.
```

- **To pass an array to a method, pass the name of the array. To pass a single element of an array to a method, simply pass the name of the array followed by the subscript of the particular element.**

- **An entire array can be passed as a parameter, making the formal parameter an alias of the original. When an entire array is passed as a parameter, a copy of the reference to the original array is passed.**

- **Arrays are passed to methods as references; therefore, the called methods can modify the element values in the caller's original arrays. Single elements of primitive-data-type arrays are passed to methods by value.**

[A POINTER is just the address of some location in memory. In Java, pointers play an important role behind the scenes in the form of references to objects. A Java variable of *object* type stores a reference to an object, which is just a pointer giving the address of that object in memory. When you use an object variable in a program, the computer automatically follows the pointer stored in that variable in order to find the actual object in memory. Since all this happens automatically, behind the scenes, you really don't have to worry much about the fact that objects are referenced through pointers.]

Example 1.

```
import apcslib.*;
public class ArrayExample1
{
    public static void main( String[] args )
    {
        double[] quantity = new double[20];
        initialize(quantity);
        System.out.print("The elements of the \"quantity\" array are:\n\n");
        showArray(quantity);
        System.out.print("\n\nThe sum of the elements of \"quantity\" array is " +
                        Format.left(getSum(quantity), 6, 2) + " .");
        System.out.print("\n-------------------------------------------------------------\n");
        for(int index=0; index<quantity.length; index++)
        {
            if(index>3 && index%4==0 ) System.out.println("\n");
            System.out.print( "quantity[" + (index) + "] = " + quantity[index] + "\t");
        }
    }

    public static void initialize( double array[] ) //"array" is an alias for "quantity"
    {
        for(int index=0; index<array.length; index++)
             array[index]=(int)(Math.random()*10000)/100.+1;
    }

    public static void showArray( double a[] ) //"a" is an alias for "quantity"
    {
        for(int index=0; index<a.length; index++)
        {
            if( index>3 && index%4==0 ) System.out.println("\n");
            System.out.print( "element[" + (index) + "] = " + a[index] + "\t");
        }
    }

    public static double getSum( double c[] ) //"c" is an alias for "quantity"
    {
        double sum = 0;
        for(int k=0; k<c.length; k++) sum += c[k];
        return sum;
    }
}
```

OUTPUT:

The elements of the "quantity" array are:

element[0] = 98.12	element[1] = 5.57	element[2] = 58.42	element[3] = 77.62
element[4] = 19.55	element[5] = 83.29	element[6] = 93.31	element[7] = 61.4
element[8] = 7.46	element[9] = 19.34	element[10] = 63.61	element[11] = 74.1
element[12] = 76.79	element[13] = 49.75	element[14] = 37.1	element[15] = 84.49
element[16] = 97.11	element[17] = 10.0	element[18] = 84.22	element[19] = 17.38

The sum of the elements of "quantity" array is 1118.63 .

---

quantity[0] = 98.12	quantity[1] = 5.57	quantity[2] = 58.42	quantity[3] = 77.62
quantity[4] = 19.55	quantity[5] = 83.29	quantity[6] = 93.31	quantity[7] = 61.4
quantity[8] = 7.46	quantity[9] = 19.34	quantity[10] = 63.61	quantity[11] = 74.1
quantity[12] = 76.79	quantity[13] = 49.75	quantity[14] = 37.1	quantity[15] = 84.49
quantity[16] = 97.11	quantity[17] = 10.0	quantity[18] = 84.22	quantity[19] = 17.38

Example 2.

```
import apcslib.*;
public class ArrayDemo1
{
    public static void main( String[] args )
    {
        double[] r = new double[8]; //r is an array ready to hold 8 double values. At this
                                    //moment it is empty.
        ArrayMethods myArray = new ArrayMethods( r );
        myArray.initialize();
        System.out.print("The elements of array \"r\" are:\n\n");
        myArray.showArray();
        System.out.print("\n\nThe sum of the elements of array \"r\" is " +
                         Format.left(myArray.getSum(), 6, 2) + " .");
        System.out.println("\n-----------------------------------------------------\n\n");
        for(int k=0; k<r.length; k++) //Notice that r has changed since it was created
        {
            if(k>3)
            {
                if(k%4==0) System.out.println("\n");
            }
            System.out.print( "r[" + (k) + "] = " + r[k] + "\t");
        }
        System.out.println("\n-----------------------------------------------------\n\n");
        System.out.print("Swapping the first and last values of the array we get:\n\n");
        myArray.swapFirstAndLast(r);
        myArray.showArray();
    }
}
```

```
public class ArrayMethods
{
    private double sum = 0, array[];

    public ArrayMethods( double m[] ) //Constructor
    { array = m; } //arrays “m” and “array” are both aliases of array “r”

    public void initialize()
    {
        for(int index=0; index<array.length; index++)
            array[index] = (int)(Math.random()*10000)/100.+1;
    }

    public void showArray()
    {
        for(int index=0; index<array.length; index++)
        {
            if( index>3 && index%4==0 ) System.out.println("\n");
            System.out.print( "element[" + (index) + "] = " + array[index] + "\t");
        }
    }

    public double getSum()
    {
        this.findSum();
        return sum;
    }

    private void findSum()
    { for(int k=0; k<array.length; k++) sum += array[k]; }

    public void swapFirstAndLast( double w[] )
    {
        double temp;
        temp = w[w.length-1];
        w[w.length-1] = w[0];
        w[0] = temp;
    }
}
```

OUTPUT:

The elements of array "r" are:

element[0] = 58.0 element[1] = 15.82 element[2] = 56.8 element[3] = 19.23

element[4] = 3.09 element[5] = 17.86 element[6] = 20.92 element[7] = 71.94

The sum of the elements of array "r" is 263.66 .

---

r[0] = 58.0 r[1] = 15.82 r[2] = 56.8 r[3] = 19.23

r[4] = 3.09 r[5] = 17.86 r[6] = 20.92 r[7] = 71.94

Swapping the first and last values of the array we get:

element[0] = 71.94 element[1] = 15.82 element[2] = 56.8 element[3] = 19.23

element[4] = 3.09 element[5] = 17.86 element[6] = 20.92 element[7] = 58.0

II. Parallel Arrays

Parallel arrays **are two arrays with corresponding elements. The following simple program — `GradeRange` — illustrates this concept.**

In the GradeRange program note that an *initializer list* can be used to instantiate an array object instead of using the new operator. The size of the array is the same as the number of items in the initializer list.

```
public class GradeRange
{
    public static void main( String[] args )
    {
        String[] grades = {"A", "A-", "B+", "B", "B-", "C+", "C", "C-",
                           "D+", "D", "D-", "F"};
        int[] cutoff = {95, 90, 87, 83, 80, 77, 73, 70, 67, 63, 60, 0};

        for( int level=0; level<cutoff.length; level++)
            System.out.println( grades[level] + "\t" + cutoff[level] );
    }
}
```

OUTPUT:

```
A   95
A-  90
B+  87
B   83
B-  80
C+  77
C   73
C-  70
D+  67
D   63
D-  60
F   0
```

Note that parallel arrays can be tricky because they can get out of synch with each other. You are usually better off creating one array that holds a single object containing all necessary information. For example, the `GradeRange` program could be changed to use a single array of objects that contain both the grade string and the numeric cutoff value.

III. Arrays of objects

Instantiating an array of objects reserves room to store references only. The objects that are stored in each element must be instantiated separately. In other words, creating the array and creating the objects that we store in the array are two separate steps. This is demonstrate in the Tunes program shown below. The programs main method creates, changes, and looks at a compact disc (CD) collection. Each CD added to the collection has a title, artist, purchase price, and number of tracks.

Also, check out the `NumberFormat` class used in classes `CDCollection` and `CD`.

```
public class Tunes
{
    public static void main( String[] args )
    {
        CDCollection music = new CDCollection();
        music.addCD("By the Way", "Red Hot Chili Peppers", 14.95, 10);
        music.addCD("Come on Over", "Shania Twain", 14.95, 16);
        music.addCD("Soundtrack", "The Producers", 17.95, 33);
        music.addCD("Play", "Jennifer Lopez", 13.90, 11);
        System.out.println( music );
        music.addCD("Double Live", "Garth Brooks", 19.99, 26);
        music.addCD("Greatest Hits", "Journey", 15.95, 13);
        System.out.println( music );
    }
}
```

```
import java.text.NumberFormat;

public class CDCollection
{
    private CD[] collection;
    private int count;
    private double totalCost;

    //Constructor - Creates an initially empty collection
    public CDCollection()
    {
        collection = new CD[100]; //Creates an array of CD's. What is a CD? (see CD class)
        count = 0;
        totalCost = 0.0;
    }

    //addCD adds a CD to the collection, increasing the size of the collection if necessary
    public void addCD( String title, String artist, double cost, int tracks)
    {
        if( count == collection.length ) increaseSize();
        collection[count] = new CD( title, artist, cost, tracks );
        totalCost += cost;
        count++;
    }

    //Doubles the size of the collection by creating a larger array
    //and copying the existing collection into it.
    private void increaseSize()
    {
        CD[] temp = new CD[collection.length * 2];
        for( int cd=0; cd<collection.length; cd++ ) temp[cd] = collection[cd];
        collection = temp;
    }
```

```
    //Returns a report describing the CD collection
    public String toString()
    {
        NumberFormat fmt = NumberFormat.getCurrencyInstance();
        String report = "************************************************************\n";
        report += "My CD collection\n\n";
        report += "Number of CD's: " + count + "\n";
        report += "Total cost: " + fmt.format(totalCost) + "\n";
        report += "Average cost: " + fmt.format(totalCost/count) + "\n";
        report += "\n\nCD List:\n\n";
        for( int cd=0; cd<count; cd++ ) report += collection[cd].toString() + "\n";
        return report;
    }
}



import java.text.NumberFormat;
import apcslib.*;

public class CD
{
    private String title, artist;
    private double cost;
    private int tracks;

    //Constructor - Creates a new CD with the specified information
    public CD( String name, String singer, double price, int numTracks )
    {
        title = name;
        artist = singer;
        cost = price;
        tracks = numTracks;
    }

    //Returns a description of this CD
    public String toString()
    {
        NumberFormat fmt = NumberFormat.getCurrencyInstance();
        String description;
        description = fmt.format(cost) + "\t" + tracks + "\t";
        description += Format.left(title, 20) + Format.left(artist, 20);
        return description;
    }
}
```

OUTPUT:

```
**********************************************************************
My CD collection

Number of CD's: 4
Total cost: $61.75
Average cost: $15.44

CD List:

$14.95   10   By the Way            Red Hot Chili Peppers
$14.95   16   Come on Over          Shania Twain
$17.95   33   Soundtrack            The Producers
$13.90   11   Play                 Jennifer Lopez

**********************************************************************
My CD collection

Number of CD's: 6
Total cost: $97.69
Average cost: $16.28

CD List:

$14.95   10   By the Way            Red Hot Chili Peppers
$14.95   16   Come on Over          Shania Twain
$17.95   33   Soundtrack            The Producers
$13.90   11   Play                  Jennifer Lopez
$19.99   26   Double Live           Garth Brooks
$15.95   13   Greatest Hits         Journey
```