Lecture 12

ch.8 – Arrays (skip 8.7 and SalesData.java example on p.484)
topics: answers to random number questions from last week –
put solutions on the board

1. Arrays

def: an array is a fixed-size collection of data elements of the
same type. Individual elements can be accessed by their index,
which is the offset from the beginning of the array. The first
element is at index zero.

Unlike other languages, where an array is simply a data
structure, in Java an array is an object, so has both data and
behavior. The data elements contained in the array can be
either primitive data or objects.

Arrays are objects in Java. There are two ways to declare and
initialize them:

In two steps:
declare the array
type[] name;  ex. int[] nGrades;  // also can be written type
name[];

allocate storage for the array
name = new type[# elements]
ex.  nGrades = new int[15]  -- an array's length will never
change.
the .length constant holds the # of elements, so nGrades.length is 15.

alternative syntax: combines declaration and allocation:
int[] nGrades = new int[15];

So far, we have allocated space for the array, but have not initialized the elements. If we don’t, Java will give them default values of zeros.

the range of indices goes from 0 to length - 1, so the first element in nGrades is nGrades[0] and the last element is nGrades[14]
so the arrayName.length is equal to 1 more than the index of the last element.

alternative syntax declares the array, allocates storage, and initializes it in one step:
int nGrades[] = {85, 72, 93, 97}; // this is called an initializer list
  // so what is nGrades[2] ?

to access one element, use its index
nGrades[0] -> 85;

Traversing the array: to visit each element, use a loop:

for (int i = 0; i < nGrades.length; i++)
    System.out.println(nGrades[i];
//this will print out 4 elements, from index 0 to index 3.

Demo use of main method parameter: String[] args
2. Passing an array as a parameter:
The variable reference is passed, so an alias is created in the formal parameter (a local variable). Since no copy of the array’s contents is passed, there is still only 1 array! This is called “passing by reference”. An alias is another name for the same object, in this case, the object is the array.

ex.  double[] dTemp = {98.5, 93.6, 87.2, 63.7}
     changeTemp(dTemp); // pass reference to the entire array

public static void changeTemp(double dT) { // dT is another name (an alias of) for the array dTemp
    dT[2] = 71.5;
} // now dTemp contains {98.5, 93.6, 71.5, 63.7}

3. Passing an array element as a parameter:
If the element is a primitive type, a copy of the value is passed. (passing by value)

double dNewTemp = convertTemp(dTemp[2]); // passes in 71.5

public static double convertTemp(double dTp) {
    dTp = 50.2; // dTemp[2] is unchanged
    return dTp;
}

    // now dNewTemp contains 50.2
4. Enhanced for loop allows for read-only of array. We can retrieve the array elements, but not change them. Type and run:

```java
/* array demo
   written by D.Bass for CS 111A,
   Fall 2005, at CCSF, 11/1/05
   Demos the enhanced for loop.
*/
public class ArrayDemo {
    public static void main(String[] args) {
        // declare and intialize an array of doubles
        double[] dTemp = {98.5, 93.6, 87.2, 63.7};
        int total = 0; // accumulator
        double average;
        System.out.print("The average of ");
        // traverse the array
        for (double temp : dTemp) {
            total += temp;
            System.out.print(temp + " ");
        }
        // calculate average
        average = total / dTemp.length;
        System.out.print("is "+average);
    }
}
```

5. Reassigning array reference variables. The reference to the original array is lost, and it is marked for garbage collection. For example:

```java
int[] firstArray = {1, 2, 3, 4, 5};

firstArray = {6, 7, 8}; // re-initialized, so original data is lost
```
Copying arrays is similar. Array reference variables hold the address in memory of the array object. When the address is copied, you have two references to the same object:

```java
int[] newArray = firstArray;
```

Any change to `newArray` is a change to `firstArray`:

```java
newArray[2] = 0;
```

Means that `firstArray` is now 1, 2, 0, 4, 5

Comparing arrays: great example on p. 482

Returning arrays from methods, note header’s return type

```java
public static double[] getArray() { // returns an array of doubles
    double[] myArray = {3.1, 4.2, 5.6};
    return myArray;
}
```
The elements of a String Array are references (addresses) of String objects:

String[] colors = {“red”, “blue”, “green”};

Colors

Sequential Search – trace SearchArray.java, p. 499