Computer Programming II Instructor: Greg Shaw

# COP 3337

**Intro to the Java *Array***

**I. Terminology**

***data structure***: a bunch of related memory locations for storing related values

***array***: a *fixed-size* data structure occupying consecutive memory locations. Stores related values of a given primitive type or Java class (from the Java Library of programmer-defined). I.e., a *list* of related values

***ArrayList*:**

a Java class that implements an abstract, object-oriented, user-friendly, v*ariable-sized* list. Although ArrayList objects are implemented as arrays, users of the ArrayList class may remain blissfully unaware of this. This is an excellent example of the OOP principle of information hiding

***array*** “***elements”***:

the components of an array, each containing a value and occupying one memory location. Each array element is just like a non-array variable (aka: *scalar* variable) except for the name, which requires a subscript (aka: index)

For this reason array elements are sometimes called *subscripted variables –* a familiar concept in math and science, although the notation is different

math/science: x0, x1, x2, ..., xn

Java: x[0], x[1], x[2], ..., x[n]

***array*** ***index***:an integer expression that tells you *which element* (i.e., the *position* of the element in the list). Aka: an array *subscript*

* Array index expressions must be enclosed in square brackets (see above)
* As with an ArrayList, the index of the first element is always 0.

1. **Advantages of Arrays (vs. ArrayLists)**
2. Prior to Java 1.5, arrays were less cumbersome when working with lists of primitive types. (This has been remedied in 1.5 with *autoboxing* and *autounboxing*.)
3. Easier to implement *multi-dimensional* arrays (tables, etc)
4. **Disdvantages of Arrays (vs. ArrayLists)**
5. Arrays are *fixed-size*, so may become full. (Although they may be resized easily, this is not done automatically as with ArrayLists)
6. Arrays may be *partially filled* (i.e., not all the elements declared may actually be used). This requires the programmer to maintain a *counter* of the number of elements actually used
7. More cumbersome insertions. When a new value is to be stored in an array, existing elements will have to be “moved down” to make room for it (unless appended to the end of the array)
8. More cumbersome deletions. When a value is removed from an array, existing elements will have to be “moved up” to fill the hole (unless deleted from the end of the array)
9. **Accessing the Individual Elements of an Array**

We access an element of an array via *index* (i.e., *subscript*) notation:

*list*[*index*]

* + *list* is the name of the array object variable
  + *index* is an integer expression that tells you *which element* (note the square brackets around the index expression)

Each array element is used *exactly* like a scalar variable. Only the name is different, in that it requires a subscript. Here is a loop that traverses an array of exactly *count* elements (i.e., elements 0 through count - 1):

**for** (**int** current = 0 ; current < count ; current++)

{

// do something here with *array[current]*

}

1. **Declaring Array Object Variables and Creating Array Objects**

*type* [] *name* = new *type*[*number-of-elements*] ;

1. *type* is the type of data stored in the array (may be any primitive type or class)
2. *name* is the name of the array object variable
3. *number-of-elements* is an integer expression indicating the array's size (i.e., number of elements).

Examples:

int [] scores = new int[35] ;

// holds up to 35 int values

boolean [] answers = new boolean[size] ;

// holds up to *size* booleans (*size* is an **int** variable)

Rectangle [] list = new Rectangle[number] ;

// holds up to *number* Rectangle object variables

Java Arrays are objects and - as with objects of any class - the object variable declaration and the creation of the actual object may be done separately. E.g.,

BankAccount [] customerList ;

.

.

.

customerList = new BankAccount[1000] ;

1. **"Overflowing the Bounds of an Array" - a Very Common Error**

Just as with an ArrayList, attempting to reference an array element that does not exist will throw an *ArrayIndexOutOfBounds* exception.

1. **The *length* Instance Variable**
2. Every array object created has an instance variable called *length* which stores the size (i.e., number of elements) of the array.
3. Instance variable *length* is commonly used in a loop when you want to "visit" every element of an array:

Example:

// create a 10-element array

double list [] = new double[10] ;

// fill array with first 10 powers of 2

for (int i = 0 ; i < list.length ; i++)

{

list[i] = Math.pow(2,i) ;

}

* Note however that *length* tells us the total number of elements in the array, i.e. the *capacity* of the array. Unlike ArrayList method *size(),* it does *not* tell us the number of elements actually used. So, if an array may be only partially filled, it is the programmer’s responsibility to keep count of the number of elements used. See *PartiallyFilled.java*

1. **Alternate Notation for Array Declarations**

There is another way to declare an array that allows us to specify the initial values stored.

In this case, we do not specify the size of the array. It is inferred by Java from the number of initial values provided.

Examples:

int [] lotto = { 14, 21, 28, 35, 42, 49 } ;

String [] colors = { “Yellow”, “Magenta”, “Cyan” } ;

BankAccount [] accounts =

{ new BankAccount(“1111111”, 15000),

new BankAccount(“2222222”, 20000),

new BankAccount(“3333333”, 12500),

new BankAccount(“4444444”, 37000) } ;

These arrays have 6, 3, and 4 elements, respectively