CPS122 Lecture: Arrays in Java

last revised January 5, 2010

Objectives:

1. To introduce arrays in Java
2. To introduce multidimensional arrays in Java

Materials:

1. Demonstration programs and projectable versions: BadReverse.java, GoodReverse.java, EvenBetterReverse.java
2. Projectable version of code excerpts illustrating array operations: sum, maximum, searching, sorting

I. Introduction

A. Python includes a number of data structures that can be used to represent collections of objects. The Java language itself only supports one (that is not found in Python); but the Java libraries include a large number of different types of collection that build on this basic type as we shall see later in the course. For now, we will focus on the type built in to the Java language - the array.

B. To understand Java arrays, it is best to mentally start from scratch.

1. Consider the following problem: we want to read in a list of 5 numbers and print them out in reverse order.

a) Clearly, we need to read all of the numbers before we can print any of them out. This means we have to store all the numbers in variables.

b) One solution would be to use 5 variables:

PROJECT BadReverse.java

DEMO

c) However, needing to have 5 distinct variables is cumbersome, and an approach like this would become essentially impossible if we were working, say, with 100 numbers, or 1000, or 10,000 !

d) To deal with situations like this, Java - like most programming languages - provides a built in data structure called an *array*. In Java, a variable is declared to be an array by following the type

name with a pair of square brackets ([]), and individual elements in the array can be referenced by following the name of the variable with a subscript enclosed in square brackets. In particular, our example could be handled as follows:

PROJECT GoodReverse.java

DEMO

Note that the complete program is now shorter than the original program - and would be much shorter if we compared programs for a larger number of values. Further, it could easily be modified to work with *any* number of numbers by changing the initial declaration of the size of the array. Every Java array has a field called length with specifies the number of elements specified when the array was created. (Note that, for arrays, this is a *field*, not a method, so no () are used.)

e) In fact, it would be easy to create a variant of this program which allows the user to specify the number of numbers when the program is run.

PROJECT EvenBetterReverse.java

DEMO

2. Recall that earlier we say that Java has two basic kinds of data types: primitive types and reference types. The latter category has two subcategories - objects and arrays. Arrays in Java can be thought of as a special kind of object); however, the formal definition of the language distinguishes them because of slight differences in the way they are used. (For example, arrays have no methods).

II. Using Arrays in Java

A. To use an array in Java, you must:

1. Declare an array variable - two alternative, but equivalent syntaxes:

< type > [] < variable name > (preferred)

Example: `int [] number;`

or

< type > < variable name > [] ("C" style declaration)

Example: `int number [];`

2. Allocate storage for the array, using new

   < variable name > = new < type > [< size >]

   (*Note:* the type used here must be the same as the type used when declaring the variable; and the size must be known at the time the array is created - it can either be an integer constant, or an integer variable or expression; in the latter case, the value of any variables at the time the array is created are what is used.)

   Example: `number = new int [5];`

   This can be combined with declaration

   < type > < variable name > [] = new < type > [< size >]

   Example: `int [] number = new int [5];`

3. You can now

   a) Refer to the array as a whole by using its name

   b) Refer to the individual elements of the array by using

      < variable name > [< subscript >]

      where < subscript > is an integer in the range 0 .. size - 1

      Examples:

      ```
      number[3]
      number[2*i+1]
      ```

      (*Note:* Java uses zero-origin indexing. An array declared with size n has elements 0 .. n - 1). So, the first element is called [0], the second [1] ...

      Note the distinction between the variable name all by itself - which stands for the *entire* array, and the variable name plus subscript, which stands for an individual *element* of the array. Operations such as arithmetic, input, and output are done on the individual elements.

      Example: If a given building is a single family home, you can address mail directly to it. If it is an apartment building, you must

specify a particular apartment by giving an apartment number as well. You can refer to the whole building for certain purposes - such as tax assessment - but most of the time you will need to refer to a specific apartment by number.

c) Refer to the number of elements in the array by

< variable name > . length

Example: `number.length`

B. One important characteristic of an array is that all of the elements of the array have the same type. The type of the elements of an array, however, can be any valid Java type.

1. A primitive type (boolean, char, int, etc.) - as in the example above

2. An object type. In this case, it is necessary not only to create the array, but also to create the individual elements of the array - since they are objects.

3. Another array type - yielding an array of arrays, or a *multidimensional array*. (We'll talk more about this later.)

C. Array initializers

1. Ordinarily, when an array is created, its elements are initialized to the default initial value for the type involved - e.g. zero for numbers, '\000' for characters, false for booleans, or `null` for reference types.

   (`null` is a Java reserved word For any reference type, `null` is the value that means the variable does not (yet) refer to anything. It is always an error to try to execute any method of a variable that is `null`)

2. It is possible, however, to specify the initial value for an array when it is declared - in which case an abbreviated notation is used that combines declaration, creation, and initialization.

   < type >[ ] < variable name > = { < expression > , < expression > ... }

   EXAMPLES

   a) An array containing of all the prime integers between 1 and 20:

   int [ ] primes = { 2, 3, 5, 7, 11, 13, 17, 19 };

b) An array of strings containing the names of the people in the first row of the room

String [] names = { --- whatever --- };

c) Typically, when we initialize an array this way, we use *constants* as the initializers. Actually, though, it is possible to use an Java expression whose value can be calculated at the point the array is declared - but we won't pursue this further.

D. Operations on Arrays

1. One typical thing to do with an array is to perform some operation on each element of the array. This is most often done with a for loop. We'll look at several examples:

a) Calculating the sum of all the elements in an array.

(1) Suppose we have an array x of doubles. To store their sum in a variable called sum, we could proceed as follows:

PROJECT

```
double sum = 0.0;
for (int i = 0; i < x.length; i ++)
    sum += x[i];
```

(2) Alternately, if we want to work with all the elements of the array - as is the case here - we could use an alternate form of the for loop (called the enhanced for loop,)

PROJECT

```
double sum = 0.0;
for (double item : x)
    sum += item;
```

(The meaning is that item is a double which should successively taken on the value of each element of the array x)

b) Finding the maximum (or minimum) valued element in an array.

Suppose we have an array x of doubles. We want to store the value of the largest element in x in a variable called max.

(1) The following is a first attempt - though it has a problem

PROJECT

```
double max = // See discussion below
for (int i = 0; i < x.length; i ++)
    if (x[i] > max)
        max = x[i];
```

The obvious problem with this solution is we do not know what initial value to give to max. How can we solve this?

ASK

(2) The following is a solution that solves our problem

PROJECT

```
double max = x[0];
for (int i = 1; i < x.length; i ++)
     if (x[i] > max)
          max = x[i];
```

Note how we start examining array elements at x[1], since we initialized max to x[0].

(3) The enhanced for loop is less applicable here, since our loop explicitly begins with element [1]; however, the correct result could still be produced by using an enhanced for, given that no harm is done comparing an item to itself.

c) Searching an array to see if a given value is present in it.

Suppose we have an array of Student objects called student, each of which has a method called getName(), and we want to see if we have a Student object for a student named "Aardvark". The following code will return the appropriate object if one exists, or null if it does not:

PROJECT

```
int i = 0;
while (i < student.length &&
      ! student[i].getName().equals("Aardvark"))
  i ++;
if (i < student.length)
  return student[i];
else
  return null;
```

(1) Notice a pattern that is characteristic of searches: the test for the loop contains *two* conditions to be tested on each iteration, which can be paraphrase as "while there is still hope of finding what we're looking for and we haven't yet found it yet ...". This relates to the fact that there are always two possible outcomes of a search: we may find what we are looking for, or we may conclude it doesn't exist.

(2) Note, too, that we test the "there is still hope of finding it" case *before* we test the "have we found what we're looking for case". Why?

ASK

The test student[i].name.equals("Aardvark") would not be legal if i were not < student.length.

d) Sorting all of the elements in the array based on their value. We'll look at one method now - a method called *bubble sort*. (There are a lot of much better methods!)

Suppose we have an array of Strings called name that we want to sort into ascending alphabetical order. The following would do the job:

PROJECT

```
for (int i = 1; i < name.length;  i ++)
    for (int j = 0; j < name.length -  i; j ++)
        if (name[j].compareTo(name[j+1]) > 0)
        {   // switch name[j] with name[j+1]
            String temp = name[j];
            name[j] = name[j+1];
            name[j+1] = temp;
        }
```

Discussion:

(1) The outer loop iterates length - 1 times

(2) Each time through the outer loop, we guarantee that the *largest* element of name[0..length - i] is placed into slot length - i - so after length - 1 iterations slots 1 .. length-1 are guaranteed to contain the correct values, which means that slot 0 does too.

(3) There are various improvements possible, which we will not discuss now.

III. Multidimensional Arrays

A. As noted earlier, the elements of an array can be of any type - including another array type. This leads to the possibility of *multidimensional* arrays.

EXAMPLE:

Suppose we were writing a computer chess game, and had created a class Piece to model individual chess pieces. Then a board could be represented as an 8 x 8 array of pieces - as follows:

```
Piece [ ] [ ] board;
board = new Piece [8] [ ];
for (int row = 0; row < 8; row ++)
    board[row] = new Piece [8];
```

We could refer to an individual piece - say the piece in row 2, column 3, by syntax like:

```
board[row][column]
```

(Note that the following syntax, used in some programming languages for accessing elements of a multi-dimensional array, is *not* legal in Java:

```
board[row, column]   // NO!!!
```

B. Actually, for initializing multi-dimensional arrays, a shorter but equivalent syntax is available

```
Piece [] [] board;
board = new Piece [8] [8];
```

This initializes board to 8 uninitialized arrays of pieces, then in turn initializes each array to be an array of 8 pieces, as desired.

C. Another important use of two-dimensional arrays is to store an image, with each element representing the brightness of a particular pixel in the image. Your first project will be based on this idea.