Blue Pelican Java
by Charles E. Cook
Lesson 18.....Arrays
Let’s suppose we need to keep grades for 400 students. Here is one way to do it:

\[ \text{int grade1} = 97, \quad \text{grade2} = 62, \quad \text{grade3} = 85, \quad \ldots \quad \text{grade400} = 76; \]

Clearly this is a tedious process for a large number of variables. Is there a better way? Yes, we should use \textbf{array variables} in this application.

Implementing our 400 variables as an array, we will use \textbf{an identical name} for all 400 variables. So how will we be able to tell them apart? We will use indices as follows (indices are sometimes called subscripts; in fact, array variables are sometimes called \textbf{subscripted variables}).

\[ \text{grade}[1] = 97; \quad \text{grade}[2] = 62; \quad \text{grade}[3] = 85; \quad \ldots \quad \text{grade}[400]; \]

Actually, this is not quite the way we do it. In reality, the indices always start at 0, so the variables would really look like this:

\[ \text{grade}[0] = 97; \quad \text{grade}[1] = 62; \quad \text{grade}[2] = 85; \quad \ldots \quad \text{grade}[399]; \]

Notice that even though we have 400 different variables in our array, the last index is 399. It is \textbf{very important} to be aware of this little quirk.
Three ways to declare and initialize an array:

Above we looked at how to initialize the various elements of an array. Let’s look now at how to declare the array … and in fact, the entire process. We will present 3 different approaches:

Before we begin to show the various approaches, let’s look first at the syntax of declaring an int array called a:

```c
int []a = ....;  //The square brackets indicate that a is to be an array. This is the
                //syntax used in most books and in contests.
int a[] = ...;   //This is a more natural way to accomplish the same thing. This is
                //the method we will use.
```

```c
int+ [ ] a = ....;
int+ a[ ] = ....
```
The first way:
```java
int a[] = new int[400];
a[0] = 97;
a[1] = 62;
a[2] = 85;
...
```

The second way:
```java
int a[] = {97, 62, 85, ...}; // This is the most popular way
```

The third way:
```java
int a[] = new int[]{97, 62, 85, ...};
```

While the above examples are for an `int` array, arrays for `double`, `String`, `char`, and `boolean` types are also possible. They are done in exactly the same way. We can even make **arrays of objects** although their initialization is slightly different. (That will be discussed later.)
We will now look at some examples of array usage, each of which will illustrate a particular feature.

**Finding the length of an array:**

`a.length` will tell us how many elements the array `a` has.

```java
double a[] = new double[7];
int lngt = a.length; //notice no parenthesis after length (it’s a state variable)
System.out.println(lngt); ///
```

```
int a[] = new int [400]; //instance variable

a.length = 400

String s = "Chris"; //method

s.length(); = s
```
Declaring and initializing on different lines:
In this example we illustrate that it’s possible to declare an array on one line and then to initialize its elements on a different line. Also, in a for loop we will take special note of the technique for cycling through all the elements of the array.

```java
int sq[] = new int[1000]; //array is only declared here...indices 0 - 999
for (int j = 0; j < sq.length; j++)
{
    sq[j] = j * j; //stores the square of each index in the element
}
```

Notice that in the code fragment `int j = 0; j < a.length` that `j` will assume values of 0 through 999. This makes a total of 1000 (0 – 999) different indices…and 1000 times through the loop.

Now let’s try to write this same code in the old fashioned way (without using arrays):

```java
sq0 = 0 * 0;
sq1 = 1 * 1;
sq2 = 2 * 2;
...
sq999 = 999 * 999;
```

This is clearly impractical and we begin to see the value of arrays.
Parallel arrays:
Consider the String array, name, and the related “parallel” int array, grade. We will cycle through a loop, inputting students’ names and corresponding grades.

```
int numStudents = 25; //this illustrates that we can use a variable to determine the length of our array
String name[] = new String[numStudents];
int grade[] = new int[numStudents];

for(int j = 0; j < numStudents; j++) {
    Scanner kbReader1 = new Scanner(System.in);
    System.out.print("Enter the student name: ");
    name[j] = kbReader1.nextLine(); //input from keyboard
    Scanner kbReader2 = new Scanner(System.in);
    System.out.print("Enter the grade: ");
    grade[j] = kbReader2.nextInt();
}
```

Because they are “associated”, the name and grade arrays are called parallel arrays.
Arrays in calculations:

We can use numeric array variables in calculations as follows:

\[
\text{average} = \frac{\text{slg}[0] + \text{slg}[1] + \text{slg}[2]}{3};
\]

This code computes the average of the first 3 elements of the \text{slg} array.
Warning:

Don’t produce an `ArrayIndexOutOfBoundsException` (an error) with improper subscripts:

```java
double zorro[] = new double[15];
zorro[14] = 37;
zorro[15] = 105; //Illegal! Index 14 is the largest possible.
zorro[0] = 209;
zorro[-1] = 277; //Illegal! Index 0 is the smallest possible.
```
Passing an array to a method:

Suppose we have the following code:

```java
char ch[] = new char[50]; // Yes, we can have character arrays
```

```java
ch[4] = 'g';
```

```java
double e = 2.718;
method1(e, ch); // call method1 (see code below) in some other class and pass our double variable and the array, ch
```

```java
System.out.println(ch[4]); // notice it's not 'g' anymore
System.out.println(e); // 2.718... unchanged
```

```
public void method1(double xxx, char myArray[])
{
    xxx = 0;
    myArray[4] = 'V';
}
```

Notice that within method1 that e was passed, but locally renamed to xxx.
Similarly, the ch array was renamed there to myArray.

a. Notice that changing xxx in method1 does not affect the e value back in the calling code.

Automatic initialization of arrays:

With numeric arrays (both *double* and *int*), all elements are automatically initialized to 0.

```java
int xyz[] = new int[2000];
System.out.println( xyz[389] ); // 0
```

The elements of a *String* array (and other object arrays) are **not** automatically initialized and will result in a *NullPointerException* when trying to reference an element that has not been specifically initialized.
Using the `split` method to produce an array:
The `split` method parses the original `String` into the separate elements of a returned `String array` using the rules of “regular expressions” (see Appendix AC) to determine the parsing delimiters.

We will not study the `split` method!!
BPJ Homework: Lesson 18

• Exercises Page 18-6 #'s 1 to 15

• Project: Array of Hope