Working With Arrays

Recall: The Array

The array is a special data structure that allows us to make large collections of data that:

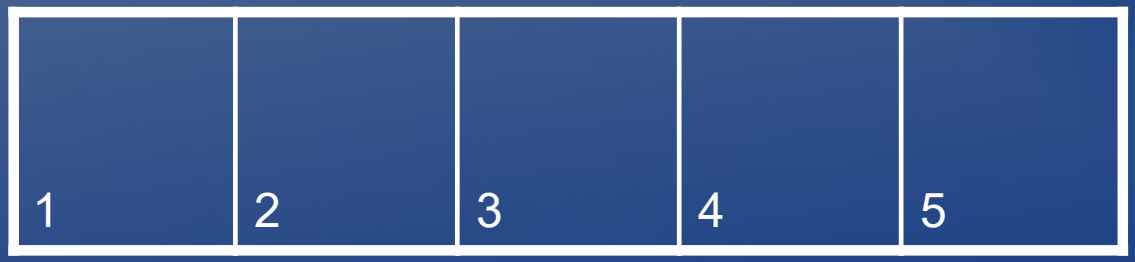
(a) are of the same data type
(i.e., int, real, string, etc...)

(b) will be used for the same purpose
(e.g., grades, names, ages, addresses, etc...)

Recall: The Array

The common way to draw or visualize the array is using a group of connected boxes. Each box in the array has a position (1st, 2nd, 3rd, etc...).

Each box is called an element of the array, and the position of each element is the index.



an array with 5 elements

Arrays in Java

An alternative way to declare arrays in one step:

```
dataType[] name = new dataType[size];
```

name – the name of the array
dataType – **`int`**, **`char`**, **`double`**, **`String`**, etc...

new – tells Java to create space in memory

size – the number of items, or elements, in array

Some Sample Array Declarations

```
// array of 10 student grades (integers)
int[] grades;
grades = new int[10];

// average temperatures for each month
double[] avgTemp;
avgTemp = new double[12];

// e-mail list for 100 members
String[] mailList = new String[100];
```

Length of an Array

After creating an array, the most important piece of information is the length of the array.

We need to know how many elements are in the array, and we must also avoid going past the boundaries of the array.

```
double[] grades = new double[10];

println(grades.length);  // outputs 10
```

Explicit Definition of an Array

It is possible to declare and define (initialize) an array at the same time, provided you already know all of the elements in advance.

```
int[] primes = {1, 2, 3, 5, 7};
```

Traversing Arrays

In many cases, we want to look at every element in the array (for input or output), from beginning to end (or reversed). This is called traversing the array, and is normally done with a FOR loop.

```
int[] primes = { 1, 2, 3, 5, 7};
for (int i = 0; i < primes.length; i++)
{
    System.out.println(primes[i]);
}
```

Array Errors - Out of Bounds

The most common error when dealing with arrays is to go "out of bounds". This is usually past the end of the array, but could be before the start as well.

This is one very good reason why we refer to the **`length`** of the array, rather than using a constant.

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

instead of

```
for (int i = 0; i < 10; i++)
```

Array Errors - Out of Bounds

A well written loop can help avoid most of these errors. If the code is more complicated, you might want to protect the array with it's own IF statement:

```
if (2*i-3>=0 && 2*i-3 < array.length)
{
   System.out.println(array[2*i - 3]);
}
```

Comparing Arrays

Arrays are complex data structures (as opposed to primitive data such as integer or char). As a result, they cannot be compared using a simple equality (==) statement.

```
int[] a = {1, 2, 3, 4, 5};
int[] b = {1, 2, 3, 4, 5};
System.out.println(a == b);  // FALSE
```

While the content of each array is the same, the arrays themselves are not equal.

Comparing Arrays

To compare arrays, each element must be compared individually and all much match. Thus the arrays must also be the same length. Use a counted (for) loop to traverse & compare each.

```
boolean same = true;
for (i = 0; i<a.length && i<b.length; i++)
{
   same = same && (a[i] == b[i]);
}
```

Comparing Arrays

```
boolean same = true;
for (i = 0; i<a.length && i<b.length; i++)
{
   same = same && a[i] == b[i];
}
```

This code is also an example of accumulating a boolean value. As soon as the comparison fails once (i.e., false), the variable 'same' will change to false and stay that way.

Accumulating boolean values can be a very useful and powerful technique for some problems.