Arrays
CS 107
Stephen Majercik

Use
To store a list of items of the same type (e.g. integers, doubles, characters) in such a way that they can be easily accessed individually.

Form
To declare and create an array (which are typically done in the same statement, though this is not necessary):

```
<type>[] <identifier> = new <type>[<number-of-items>];
```

where type indicates the type of the items in the array, `[]` indicates that it is an array of these items, identifier is the name of the array, `new` allocates the space for the array, and `number-of-items` indicates how many items of that type the array can hold. Although number-of-items is often declared as a named constant, this is not necessary in Java. You can get an integer value from the user that indicates the size of the array and use the variable containing that value as `<number-of-items>`. See examples below. The items in numeric arrays are automatically initialized to zero. It is also possible to give non-zero initial values to the items in the array when you declare it by setting the declared array equal to a list of the values, separated by commas, within curly braces:

```
<type>[] <identifier> = { <val-1, val-2, ..., last-val> };
```

Note that this automatically indicates the size of the array as well (the number of values in the series of initialization values). If you want to initialize all the values in a number array to a nonzero value other than zero, and you don't want to, or can't, list that value out repeatedly, you will need to use a loop (see below for examples).

Using an Array
Each position in an array has an *index* associated with it. Indices are zero-based, so the index of the first item is 0, the index of the second item is 1, etc. This means that if the array contains 6 items, the index of the last item is 5.

An item in an array is accessed by specifying its index. For example, the following array declaration creates an array that can hold four `doubles` and the value of each item is initially 0.0. The name of the array indicates that these are barometric pressures, but note that, as is the case with regular variables, the name is for the convenience of the programmer and anyone else reading the program. We could call the array `partyFavors` and the program would still work, although I would deduct points for not using *accurately* descriptive identifiers. `;-)`

```
final int MAX_NUM_BAROMETER_READINGS = 4;
double[] barometricPressure = new double[MAX_NUM_BAROMETER_READINGS];
```

The following would be equivalent, assuming the user input the value 4:

```
System.out.print("How many barometer readings do you have?  ");
int numBarometerReadings = r.readInt();
r.readLine();
double[] barometricPressure = new double[numBarometerReadings];
```

To access the item that has index 2, we write:

```
barometricPressure[2]
```

and this can be used on the left-hand side of an assignment statement, if we are assigning a value to that item in the array:

```
barometricPressure[2] = 29.5;
```

or on the right-hand side of an assignment statement, if we are using that value some calculation:

```
double someKindOfVariable = barometricPressure[2] * 2;
```

Typically, however, we don't use constant numbers to "index into the array" to access a particular item. Usually, we want to go through the array and do something with each item. And here's where the power/convenience of arrays becomes clearer. We can use a *variable* to specify the index. In other words, if I have the following:

```
barometricPressure[i]
```

and `i` has the value 2, this will access the item at index 2. And if I change the value of `i` to 3, `barometricPressure[i]` will now access the item at index 3. So, if I want to go through the entire array and do something with each item, I can accomplish this by declaring an integer variable that I will use as the index, and setting up a loop that makes this variable take on all possible values of the index (i.e. from 0 up to 1 less than the number of items in the array). Suppose, for example, I want to initialize all the barometric pressures to 30.0. I can do this as follows. Note that index variables are often (though not always) just called `i`, `j`, or `k`.

```
int i = 0;
while (i < barometricPressure.length) {
   barometricPressure[i] = 30.0;
   ++i;
}
```

Notice that an array knows its own length (or number of items) and that information is accessible through `<array-id>.length`. Also, notice that, because of the zero-based array indexing, `i` must take on values from `0` to `(barometricPressure.length - 1)`. This may be confusing at first, but the alternative is just as bad:

```
int i = 1;
while (i <= barometricPressure.length) {
   barometricPressure[i-1] = 30.0;
   ++i;
}
```

Here you need to remember to subtract 1 from the index.

Now, suppose I want to double each barometric pressure:

```
int i = 0;
while (i < barometricPressure.length) {
   barometricPressure[i] = barometricPressure[i] * 2;
   ++i;
}
```

Now, suppose I want to print them out:

```
int i = 0;
while (i < barometricPressure.length) {
   System.out.println(``Barometric pressure '' + (i+1) + `` = '' +
                         barometricPressure[i]);
   ++i;
}
```

Notice that I have added 1 to i so that the output is not 0-based, i.e. it reads:

```
Barometric pressure 1 = 60.0
Barometric pressure 2 = 60.0
Barometric pressure 3 = 60.0
Barometric pressure 4 = 60.0
```

rather than:

```
Barometric pressure 0 = 60.0
Barometric pressure 1 = 60.0
Barometric pressure 2 = 60.0
Barometric pressure 3 = 60.0
```

Most programmers favor `for`-loops when going through an array because all the loop control information is in one place, the `for`-loop header. For example, the last `while`-loop above, would be written as follows as a `for`-loop:

```
for (int i = 0 ; i < barometricPressure.length ; ++i ) {
   System.out.println(``Barometric pressure '' + (i+1) + `` = '' +
                        barometricPressure[i]);
}
```

I'll discuss `for`-loops in more detail in class.