CS125 : Introduction to Computer Science

Lecture Notes #8
One-Dimensional Arrays

Lecture 8 : One-Dimensional Arrays

The need for arrays

An example we *can* do, given our knowledge so far: “For a given student, read 10 exam scores and print out the total of those exam scores.” In this case, we didn’t need to save the exam scores; we could just add them to the same variable one by one:

```
int total = 0;
for (int i = 1; i<=10; i++)
{
   System.out.print("Enter score #" + i + ": ");
   total = total + Keyboard.readInt();
}
System.out.println("Total is " + total + ".");
```

New problem: “For a given student, read 10 exam scores, print their total, and *then* print each exam score as well.” Now, the above will not do! We haven’t saved the values so we have no way to print them out. We know their total, but there is no way to obtain each of the ten individual exam scores from that.

Since the above doesn’t work, we need to try something else. One thing we *could* do is to save the exam scores as we read them. Since there are ten exam scores, we’d have to save them in 10 separate variables, and then later, print those 10 variables out one by one.

```
int score1, score2, score3, score4, score5, score6, score7, score8,
                                              score9, score10, total;

System.out.print("Enter score #1: ");
score1 = Keyboard.readInt();
System.out.print("Enter score #2: ");
score2 = Keyboard.readInt();
System.out.print("Enter score #3: ");
score3 = Keyboard.readInt();
          ...  // similar code for scores 4-9
System.out.print("Enter score #10: ");
score10 = Keyboard.readInt();

total = score1 + score2 + score3 + score4 + score5 + score6 +
          score7 + score8 + score9 + score10;

System.out.println("Total is " + total + ".");

System.out.println("Score #1 is " + score1 + ".");
System.out.println("Score #2 is " + score2 + ".");
System.out.println("Score #3 is " + score3 + ".");
          ...  // similar code for scores 4-9
System.out.println("Score #10 is " + score10 + ".");
```

but that has two problems:

- We have to write out the prompt and input statments 10 separate times. There is no way to use a loop, since each input requires a slightly different variable name. Likewise, we need 10 print statements – we cannot use a loop, since each print statement is printing a slightly different variable name.
- This repetitive work is bad enough for only 10 variables. What if there were 100? 1000? What if we didn't know in advance how many we would need?

What we would like to have is a way of writing the previous code, but without the two problems we describe above. Note that all the variables `score1` through `score10` are very similar in name – the only thing different is the number at the end. What if we had some way to make the computer recognize this? What if we could simply use the variable name "`score`" along with some number, and the computer could easily put the name "`score`" and the number together, to get the appropriate full variable name, such as `score1` or `score4` or `score8`?

Arrays

Our solution will be very similar to that idea. What we will discuss today is a computer language tool known as an *array*. An array is essentially a collection of variables – called *cells* in this context – which are all of the same type and are all part of one collection with a single name. For example:

```
        -------------------------------
        |    |    |    |    |    |    |
arr     |    |    |    |    |    |    |
        |____|____|____|____|____|____|

          0    1    2    3    4    5
```

Above we see an array, whose name is `arr`. There are six cells in this array, each with an associated integer called an *index*, or less commonly, a *subscript*. The indices of our six cells are 0, 1, 2, 3, 4, and 5, respectively. Every cell will have an index, the set of indices is always a range of consecutive integers, and always (in Java) starts at 0. So, an array with 10 cells would have them indexed 0 through 9, and array with 148 cells would have them indexed 0 through 147, and so on. Above, our array of six cells has them indexed 0 through 5.

An array is created with a line such as the following:

```
int[] arr = new int[6];
```

This line tells the computer to create an array named `arr`, and to have that new array contain 6 cells (so the indices are therefore 0 through 5), and to have those cells be of type `int`. In general, the form of such a statement is:

```
SomeType[] arrayName = new SomeType[ExpressionForNumberOfCells];
```

where `SomeType` is whatever type you want (`int` in the example above), `arrayName` is a variable name of your choosing (`arr` in the example above), and `ExpressionForNumberOfCells` is some expression that evaluates to a non-negative integer (in the example above, our integer literal 6 evaluates to a non-negative integer, namely, 6, so that is fine as well). So, another example of an array creation would be the third line in the following code snippet:

```
int a = 10;
int b = 5;
double[] results = new double[a+b];  // array created
```

In that example, the array cells are of type `double`, the array name is `results`, and there are 15 cells total, and so those cells would have indices 0 through 14. We will explain a bit more about this syntax – including why it looks like an assignment statement – in the coming lectures, but for now, you at least know what line you would type to create an array.

You refer to a particular cell in a particular array via a combination of the name and an index. Each name can only refer to one array – that is, just as you can't have two `int` variables both named `theValue`, you can't have two arrays both named `arr`. A name will identify exactly one particular array. And then the index uniquely identifies a cell within that array. So, the combination of the two – name *and* index together – uniquely identifies an array cell.

You indicate this combination of name and index using square brackets (`[  ]`). You have the following form:

```
arrayName[indexExpression]
```

where `arrayName` is the name of your array, and `indexExpression` is some expression that evaluates to an index of the array. For example:

```
arr[0]       // this expression gives you cell 0 of
             //   the array in the above picture
arr[1]       // this expression gives you cell 1 of
             //   the array in the above picture
arr[4]       // this expression gives you cell 4 of
             //   the array in the above picture
arr[2+3]     // this expression gives you cell 5 of
             //   the array in the above picture
arr[2*(x-r)] // if x holds 3 and r holds 1, this expression
             //   gives you cell 4 of the array in the above
             //   picture
arr[7]       // this expression will make your program
             //   crash when it runs, since 7 is not an
             //   index in the array named ``arr''
```

You can use this name-index combination as a variable name, to either write to or read from. So, just as you could write statements using single variables such as the following:

```
int a;
a = 2;                 // assigning the variable
System.out.println(a);  // printing the variable
```

you can write similar statements such as:

```
int[] arr = new int[6];
arr[4] = 17;                // assigning the array cell
System.out.println(arr[4]); // printing the array cell
```

The first line will create the array:

```
      ------------------------------
      |    |    |    |    |    |    |
arr   |    |    |    |    |    |    |
      |____|____|____|____|____|____|

        0    1    2    3    4    5
```

The second line will write the value 17 into cell 4 of that array:

```
     ------------------------------
     |    |    |    |    |    |    |
arr  |    |    |    |    | 17 |    |
     |____|____|____|____|____|____|

        0    1    2    3    4    5
```

and the third line will read the value 17 from cell 4 of that array and print that value 17 to the screen.

Likewise, the following statements:

```
arr[0] = 13;
arr[1] = 128;
arr[2] = -10;
arr[3] = 99;
arr[5] = 0;
```

would then write values into the remaining cells of the array.

```
     ------------------------------
     |    |    |    |    |    |    |
arr  | 13 |128 |-10 | 99 | 17 | 0  |
     |____|____|____|____|____|____|

        0    1    2    3    4    5
```

Solving our earlier problem

So now, let's solve that earlier problem. Instead of having 10 different integer variables, let's simply create one array with 10 cells:

```
int[] scores = new int[10]; // indices 0 through 9
```

Our code for the prompting and inputting would now look like this:

```
System.out.print("Enter score #1: ");
scores[0] = Keyboard.readInt();
System.out.print("Enter score #2: ");
scores[1] = Keyboard.readInt();
System.out.print("Enter score #3: ");
scores[2] = Keyboard.readInt();
        ... // and so on for cells 3 through 9
```

except now, since the index can be an expression, we *can* use a loop!

```
for (int i = 1; i <= 10; i++)
{
   System.out.print("Enter score #" + i + ": ");
   scores[i-1] = Keyboard.readInt();
}
```

Then, we only need to take care of totalling and printing and we're all set. The finished code snippet is as follows:

```
int[] scores = new int[10];
int total = 0;
for (int i = 1; i <= 10; i++)
{
   System.out.print("Enter score #" + i + ": ");
   scores[i-1] = Keyboard.readInt();
}

for (int i = 1; i <= 10; i++)
   total = total + scores[i-1];
System.out.println("Total is " + total + ".");

for (int i = 0; i < 10; i++)
   System.out.println("Score #" + (i+1) + " is " + scores[i] + ".");
```

Note that the loops can either run from 1 through 10, or from 0 through 9 – you only need to alter the index expression in the body of the loop to take the particular loop range into account.

The two problems we had earlier have now gone away:

- We no longer need to repeat the same snippet of code 10 times; instead, we can use a loop to have a variable `i` incremement each time through the loop, and use an expression involving `i` for the array index.
- If we wanted to change this to 100, or 1000 scores, rather than just 10, all we would need to do is change the "10"s above to a different number.

In fact, since the size of the array can also be an expression, it is possible to have read in the size as user input. The following program will work for any number of scores.

```
public class ScorePrinting
{
   public static void main(String[] args)
   {
      int numberOfScores;
      System.out.print("How many scores are there?: ");
      numberOfScores = Keyboard.readInt();

      int[] scores = new int[numberOfScores];
      int total = 0;

      for (int i = 1; i <= numberOfScores; i++)
      {
         System.out.print("Enter score #" + i + ": ");
         scores[i-1] = Keyboard.readInt();
      }

      for (int i = 1; i <= numberOfScores; i++)
         total = total + scores[i-1];
      System.out.println("Total is " + total + ".");

      for (int i = 0; i < numberOfScores; i++)
         System.out.println("Score #" + (i+1) + " is " + scores[i] + ".");
   }
}
```

The `length` variable

Every array has a built-in `length` variable that tells you how many cells the array has. You access this `length` variable by using the array name, followed by a period, followed by the name `length`:

```
arr.length
```

This `length` value is initialized automatically whenever we create an array. So, as soon as we executed a statement such as:

```
int[] scores = new int[10];
```

then from that point on, the expression `scores.length` will always evaluate to 10, since that is how many cells the array has. We could have written the previous code slide as follows:

```
public class ScorePrinting
{
   public static void main(String[] args)
   {
      int numberOfScores;
      System.out.print("How many scores are there?: ");
      numberOfScores = Keyboard.readInt();

      int[] scores = new int[numberOfScores];
      int total = 0;

      for (int i = 1; i <= scores.length; i++)
      {
         System.out.print("Enter score #" + i + ": ");
         scores[i-1] = Keyboard.readInt();
      }

      for (int i = 1; i <= scores.length; i++)
        total = total + scores[i-1];
      System.out.println("Total is " + total + ".");

      for (int i = 0; i < scores.length; i++)
         System.out.println("Score #" + (i+1) + " is " + scores[i] + ".");
   }
}
```