

Chapter 3

Arrays

Arrays represent an ordered sequence of values. An array in Java can only hold values of one particular type, which is indicated at declaration.

- Arrays are not quite the same as lists (though Python lists also use [] syntax). A list can change in length easily, but might take much longer to perform value lookups, because each element is chained to the next. An array uses a fixed layout and so looking up values can be much faster (just jump to the correct location), but changing the length (adding or removing elements in the middle) can take a lot of time to perform all the copying.
 - An **array type** does not include the length of the array (or any of the dimensions, when working with multi-dimensional arrays like `int[][]`).
 - While the initial length of an array can be decided at run-time (e.g., ask for a number, make an array that long), an array value does not change in length. If you want a longer one, you must create a new longer array value, copy everything over, and then use the extra space.
-

Declaration

To declare an array, we again give a type and an identifier to announce to Java that something of this type should now exist, with this name. To create an array of type T, the array type is created by appending [] to the end of the type: `T[]`. Other examples would be `int[]`, `double[]`, `int[][]`, and so on. Here are some declarations:

```
int[ ] xs;  
double[ ] samples;  
String[ ] names;
```

I tend to make names like `xs`, `ys`, `zs`. They should be pronounced "exes, why's, zees", and are supposed to represent multiple x values (whatever x is), multiple y values, and so on.

Your Turn!

- Try declaring a few arrays, and see if your code still compiles. We'll give them values next.
-

Instantiation – two options

If you know the specific values to start out with, you can place them in curly braces { }. This is only an option, though, at declaration time:

```
int[] xs = {5, 6, 7, 8, 9};  
double[] ds = {2.5, 3.5, 4.5, 5.5, 6.0 };  
int[][] yss = { { 1, 2, 3}, {5, 6, 7, 8}, {4} };
```

If you don't want to or can't supply those values initially, we can instead specify the dimension(s), and get default values (of zero, false, space-character, or null for Classes):

```
int[] xs;                // declaration can now be separate.
xs = new int[5];         // xs now holds {0,0,0,0,0}.
ys [] [] = new int[2][3]; // ys now holds {{0,0,0},{0,0,0}}.
bs[] = new boolean[3];  // bs now holds {false, false, false}.
```

This style is often used in tandem with a for-loop that then follows up and systematically updates the value at each location. We will cover this in a couple of sections.

Your Turn!

- Now that you have a couple of arrays declared, try giving them values using both means of instantiation. Again, just make sure it compiles. If you try `System.out.println(xs)`, you will see essentially the array's address and not its contents. Take a peek back at the "for-each loop" section for more syntax on writing for-loops for arrays. We're going to access the elements next!

Accessing, Updating

Java uses square brackets `[]` to describe a particular indexed location in an array. In the same way that a variable can show up on the left and right sides of an assignment and mean a location of storage (on the left) and an expression to look up the value (on the right), we can do this with arrays and array syntax.

- Java arrays use zero-based indexes. If there are 5 elements in the array, the available indexes are 0,1,2,3,4.
- A value in an array is found by giving the name of the array, an open bracket, an index number (integer that happens to be a valid index), and a close bracket.
- Any integer value can be used as the index, whether it's a literal, a variable, or some other expression.

The length of an array is found by:

```
arrayName.length
```

Note that this length is how many spots are in the array, which is one larger than the largest index.

```
int[] xs = {2,4,6,8,10};
//access elements in the array:
int midVal = xs[2];
System.out.println("the first value in xs is : " + xs[0] );

// change the value at one location in the array:
xs[3] = 33;

//the array now contains {2,4,6,33,10}.
System.out.println ("The xs array is " + xs.length + " spots long.");
```

Your Turn! Now we finally see how to access elements and print them!

- Try printing out a couple of different elements in your arrays.
- Make sure one of your arrays uses at least two dimensions, such as `int[][] zs = {{2,4,6},{1,3,5}}`. Try accessing individual values in it and also modify them.

Loops and Arrays

Now that we know about arrays and how to access them, and we know about loops and how they let us systematically step through a range of numbers, we can use the loop to use each index in separate

iterations, allowing us to visit each element in an array. The following example creates an array, specifies how long it is, uses a loop to fill in each spot (updates them) according to some pattern, and then prints them out (accesses the elements).

```
int[] xs = new int[15];

for(int i=0; i< xs.length; i++) {
    xs[i] = i*i;
}

System.out.println("Some square numbers: ");
for (int i=0; i<xs.length; i++) {
    System.out.print(xs[i] + " ");
}
```

Your Turn!

- Print out all the elements in your arrays you've created. How will you handle the multiple dimensions?
- Use a Scanner and ask the user for 10 integers, storing them in an array. Print them back out.
- Generalize the Scanner example to be however many numbers they first tell you they want to enter. Does this affect where you declare your array? What about where you instantiate it?
- Try calculating the sum of an array of integers – what other storage might you need? What type should it be?
- Choose one of your arrays, and print it out its contents as we've done before, such as {2,4,6,1,3,5,7}, but based on whatever values happen to be in there.
- Report both the *index* of the maximum value in the array, as well as the *value* of the maximum value in the array.