**Accessing Elements in Arrays**

The two most primitive Array operations are writing elements into them, and reading elements from them. All other Array operations are built on top of these two primitive operations.

Writing Items into an Array

To put a DVD into the Array, we need to decide which of the 15 places we'd like it to go in. Each of the places is identified using a number in the range of 0 to N - 1. The 1st place is 0, the 2nd place is 1, the 3rd place is 2... all the way up to the 15th place, which is 14. We call these numbers that identify each place **indexes**.

Let's put the DVD for *The Avengers* into the eighth place of the Array we created above.

|  |
| --- |
| // Firstly, we need to actually create a DVD object for The Avengers.  DVD avengersDVD = new DVD("The Avengers", 2012, "Joss Whedon");  // Next, we'll put it into the 8th place of the Array. Remember, because we  // started numbering from 0, the index we want is 7.  dvdCollection[7] = avengersDVD; |

And that's it. We've put the DVD for The Avengers into our Array! Let's put a few more DVD's in.

|  |
| --- |
| DVD incrediblesDVD = new DVD("The Incredibles", 2004, "Brad Bird");  DVD findingDoryDVD = new DVD("Finding Dory", 2016, "Andrew Stanton");  DVD lionKingDVD = new DVD("The Lion King", 2019, "Jon Favreau");  // Put "The Incredibles" into the 4th place: index 3.  dvdCollection[3] = incrediblesDVD;  // Put "Finding Dory" into the 10th place: index 9.  dvdCollection[9] = findingDoryDVD;  // Put "The Lion King" into the 3rd place: index 2.  dvdCollection[2] = lionKingDVD; |

Notice that we put The Incredibles into the Array at index 3. What happens if we now run this next piece of code?

|  |
| --- |
| DVD starWarsDVD = new DVD("Star Wars", 1977, "George Lucas");  dvdCollection[3] = starWarsDVD; |

Because we just put Star Wars into the Array at index 3, The Incredibles is no longer in the Array. It has been overwritten! If we still have the incrediblesDVD variable in scope, then the DVD still exists in the computer's memory. If not though, it's totally gone!

### Reading Items from an Array

We can check what's at a particular Array index.

|  |
| --- |
| // Print out what's in indexes 7, 10, and 3.  System.out.println(dvdCollection[7]);  System.out.println(dvdCollection[10]);  System.out.println(dvdCollection[3]);  // Will print:  // The Avengers, directed by Joss Whedon, released in 2012  // null  // Star Wars, directed by George Lucas, released in 1977 |

Notice that because we haven't yet put anything at index 10, the value it contains is null. In other languages, such as **C**, the Array slot could contain completely random data. Java always initializes empty Array slots to null if the Array contains *objects*, or to default values if it contains *primitive types*. For example, the array int [] would contain the default value of 0 for each element, float[] would contain default values of 0.0, and bool[] would contain default values of false.

Writing Items into an Array with a Loop

We commonly use a loop to put lots of values into an Array. To illustrate this, let's go to another example. This time, we're going to create an Array of ints and put the first 10 square numbers into it.

|  |
| --- |
| int[] squareNumbers = new int[10];  // Go through each of the Array indexes, from 0 to 9.  for (int i = 0; i < 10; i++) {  // We need to be careful with the 0-indexing. The next square number  // is given by (i + 1) \* (i + 1).  // Calculate it and insert it into the Array at index i.  int square = (i + 1) \* (i + 1);  squareNumbers[i] = square;  } |

Reading Items from an Array with a Loop

We can also use a loop to print out everything that's in the Array.

|  |
| --- |
| // Go through each of the Array indexes, from 0 to 9.  for (int i = 0; i < 10; i++) {  // Access and print what's at the i'th index.  System.out.println(squareNumbers[i]);  }  // Will print:  // 1  // 4  // 9  // 16  // 25  // 36  // 49  // 64  // 81  // 100 |

One last thing worth knowing now is that there's a more elegant way of printing out the values of an Array—a variant of the for loop, commonly referred to as a "for each" loop.

|  |
| --- |
| // For each VALUE in the Array.  for (int square : squareNumbers) {  // Print the current value of square.  System.out.println(square);  }  // Prints exactly the same as the previous example. |

You'll probably agree that this code is a lot simpler to read. We can use it whenever we don't need the index values. For actually writing the squares into the Array, it wouldn't have worked because we needed to work with the actual index numbers. You don't have to use a "for each" loop when you're starting out, but we recommend you become comfortable with it before interviews. Simple, elegant code is good code!