One-dimensional Arrays (Part 1)

Lesson Contents

Part 1

1. What is an array?

2. Array indexing

3. Ways of declaring an array reference variable, instantiating an array, and initializing an array

4. The public final *length* instance variable

5. How and when to use a *for* loop to traverse an array

Part 2

6. How and when to use a *for-each* loop to traverse an array

7. Array references as method arguments

8. Arrays of object references

9. Best and worst case analysis of insertion and removal operations

1. What is an array?

An array is a data structure used to maintain a list of elements of the same type. An array is implemented as a sequence of contiguous blocks of memory, one block for each element in the list. (See Fig. 1.) Once an array is created, its length is fixed.

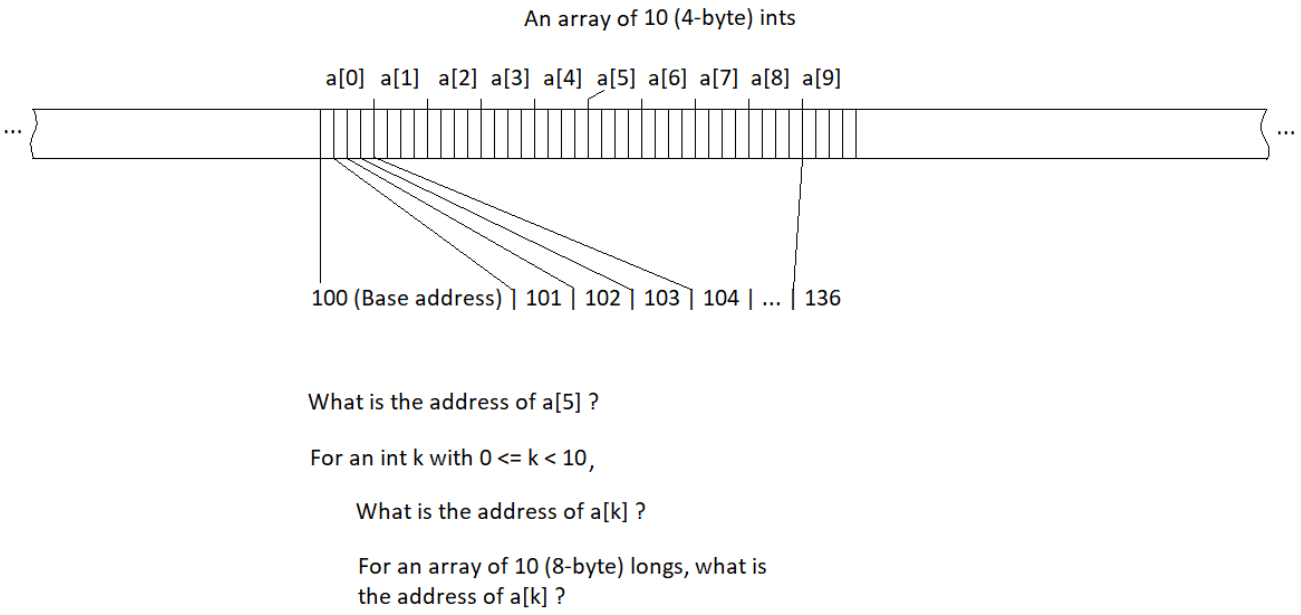

Fig. 1

2. Array indexing

The Java syntax for creating and accessing arrays does not involve specifying memory addresses or calculating the byte offsets of array elements. For an array named *a*, the first element is referred to as *a*[0], the second as *a*[1], and so on. (See Fig. 1.) So if the array *a* contains *n* elements, the final element is referred to as *a*[*n*-1]. Attempting to reference an element outside the range [0, n-1] will generate an *ArrayIndexOutOfBoundsException*.

3. Ways of declaring an array reference variable, instantiating an array, and initializing an array

Arrays are objects, and so they are identified with reference variables. For example,

int[] a;

declares a reference variable *a* that can hold a reference to an array of ints.

int[] a, b, c;

declares 3 reference variables *a, b,* and *c,* each of which can hold a reference to an array of ints.

To create an array, we can use the *new* operator, as when creating a class instance. For example,

a = new int[10];

creates an array of 10 ints, and stores the reference to it in the variable *a.*

The above two lines can be combined into

int[] a = new int[10];

which has the exact same effect.

The notation

int a[] = new int[10];

is equivalent.

Multiple arrays with elements of the same type can be instantiated in one statement as in :

int[] a = new int[10], b = new int[20], c = new int[30];

Once an array is created in any of the above ways, each of its elements will have a default value. For arrays of type int, long, float, or double, the default value is 0. For arrays of type boolean, the default is *false*. For arrays of object references, the default value is *null*. Once the array is created, its length is fixed.

To initialize an array, with values other than the defaults, the [] notation can be used on the left hand side of assignments. For example,

```
a[0] = 2;
a[1] = 3;
a[2] = 5;
a[3] = 7;
a[4] = 11;
a[5] = 13;
a[6] = 17;
a[7] = 19;
a[8] = 23;
a[9] = 29;
```

will store the first 10 prime numbers in positions a[0] through a[9]. In cases like this, when the length of the array is not too large, the following notation can be used to instantiate and initialize the array in one statement.

int[] primes = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29}; // Notice that the *new* operator is not used.

For larger arrays, a loop can be used to initialize the array. (See section 5.)

4. The public final *length* instance variable

Array instances have a public final instance variable called *length* that is equal to the array length, which is constant. So for example, given the *primes* array as defined above,

System.out.println(primes.length);

will print 10.

What would be the result of the following statement?

System.out.println(primes[primes.length]);

5. How and when to use a *for* loop to traverse an array

A for loop can always be used to traverse an array, for initializing, reading, or updating the elements. For example,

```
for (int i = 0;  i < a.length; i++) {
  a[i] = (int)(Math.random() * 101);  // fill the array with random integers
}
```

will set each element to a random integer in the range [0,100].

The array can be traversed without modifying its contents:

```
// display the array contents
//
for (int i = 0;  i < a.length; i++) {
  System.out.print(a[i] + " ");
}

// Find the largest element in the array
//
int max_i = 0; // Index of the largest element found so far
for (int i = 0;  i < a.length; i++) {
  if (a[i] > a[max_i]) max_i = i;
}
// a[max_i] is the largest value in the array.
```