Object Oriented Programming

#7 Array

Hilmy A. Tawakal

October 23, 2019

Table of contents

Background

Suppose you write a program that reads a sequence of values and prints out the sequence, marking the largest value, like this:

```
32
54
67.5
29
35
80
115 <= largest value
44.5
100
65
```

Storing a sequence of values

- Therefore, the program must first store all values before it can print them.
- Could you simply store each value in a separate variable? (value1,value2,...,value10)
- However, such a sequence of variables is not very practical to use. You would have to write quite a bit of code ten times, once for each of the variables.
- In Java, an array is a much better choice for storing a sequence of values of the same type.

Declaring Array

1. Here is the declaration of an array variable of type double[]

   ```
   double[] values;
   ```

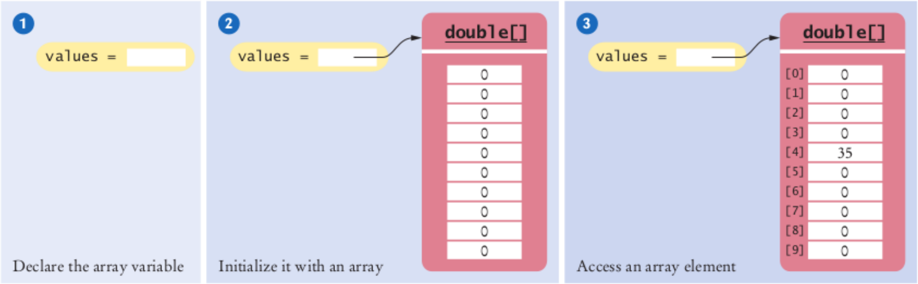
2. When you declare an array variable, it is not yet initialized. You need to initialize the variable with the array:

   ```
   double[] values = new double[10];
   ```

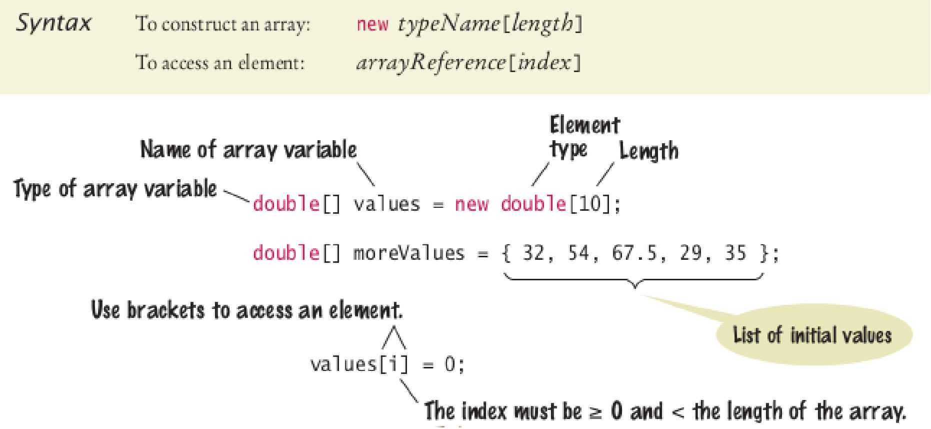
3. To access a value in an array, you specify which “slot” you want to use. That is done with the [] operator:

   ```
   values[4] = 35;
   ```

Declaring Array



Array syntax



Syntax To construct an array: new *typeName*[*length*]

To access an element: *arrayReference*[*index*]

```
double[] values = new double[10];

double[] moreValues = { 32, 54, 67.5, 29, 35 };

values[i] = 0;
```

Element type
Length
Name of array variable
Type of array variable
List of initial values
Use brackets to access an element.
The index must be ≥ 0 and < the length of the array.

Array syntax

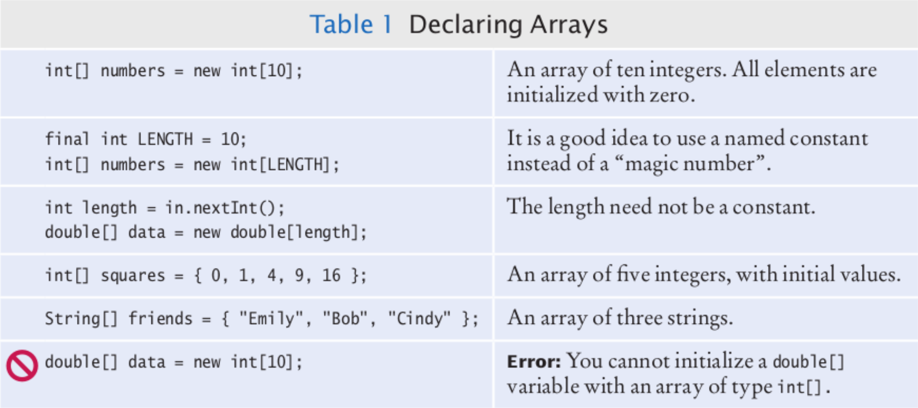


Table 1 Declaring Arrays

`int[] numbers = new int[10];`	An array of ten integers. All elements are initialized with zero.

`final int LENGTH = 10;`
`int[] numbers = new int[LENGTH];`
	It is a good idea to use a named constant instead of a “magic number”.

`int length = in.nextInt();`
`double[] data = new double[length];`
	The length need not be a constant.

`int[] squares = { 0, 1, 4, 9, 16 };`	An array of five integers, with initial values.
`String[] friends = { "Emily", "Bob", "Cindy" };`	An array of three strings.
🚫 `double[] data = new int[10];`	**Error:** You cannot initialize a `double[]` variable with an array of type `int[]`.

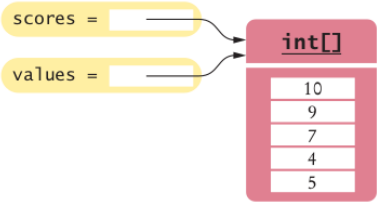
Array References

- When you copy an array variable into another, both variables refer to the same array

```
int[] scores = { 10, 9, 7, 4, 5 };
int[] values = scores; // Copying array reference
```

- You can modify the array through either of the variables:

```
scores[3] = 10;
System.out.println(values[3]); // Prints 10
```



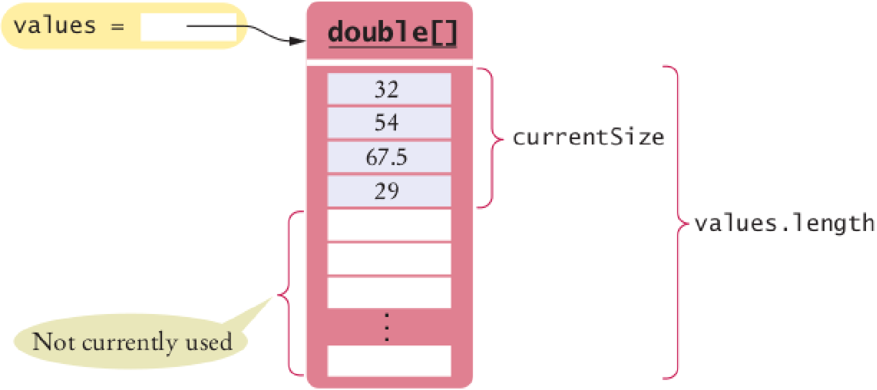
Partially Filled Array

- An array cannot change size at run time.
- This is a problem when you don't know in advance how many elements you need.
- In that situation, you must come up with a good guess on the maximum number of elements that you need to store.
- For example, we may decide that we sometimes want to store more than ten elements, but never more than 100:

```
final int LENGTH = 100;
double[] values = new double[LENGTH];
```

- In a typical program run, only a part of the array will be occupied by actual elements
- We call such an array a **partially filled array**.
- You must keep a *companion variable* that counts how many elements are actually used

Partially Filled Array



Partially Filled Array

- The following loop collects inputs and fills up the values array:

```
int currentSize = 0;
Scanner in = new Scanner(System.in);
while (in.hasNextDouble())
{
  if (currentSize < values.length)
  {
    values[currentSize] = in.nextDouble();
    currentSize++;
  }
}
```

- At the end of this loop, *currentSize* contains the actual number of elements in the array.
- Note that you have to stop accepting inputs if the *currentSize* companion variable reaches the array length.

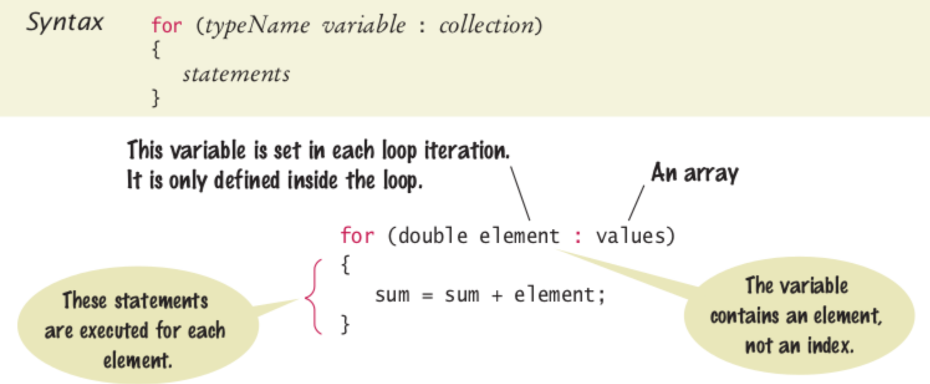
The Enhanced for Loop

- Often, you need to visit all elements of an array
- The *enhanced for loop* makes this process particularly easy to program.
- Here is how you use the enhanced for loop to total up all elements in an array named values :

```
double[] values = . . .;
double total = 0;
for (double element : values)
{
  total = total + element;
}
```

- The loop body is executed for each element in the array values.
- At the beginning of each loop iteration, the next element is assigned to the variable element.
- Then the loop body is executed.

The Enhanced for Loop



Syntax

```
for (typeName variable : collection)
{
   statements
}
```

```
for (double element : values)
{
   sum = sum + element;
}
```

This variable is set in each loop iteration. It is only defined inside the loop.
An array
These statements are executed for each element.
The variable contains an element, not an index.

Filling

- This loop fills an array with squares (0, 1, 4, 9, 16, ...). Note that the element with index 0 contains 0^2, the element with index 1 contains 1^2 , and so on.:

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

Sum and Average Value

- When the values are located in an array, the code looks much simpler:

```
double total = 0;
for (double element : values)
{
  total = total + element;
}
double average = 0;
if (values.length > 0) { average = total /
    values.length; }
```

Maximum and Minimum

- Here is the implementation of that algorithm for an array:

```
double largest = values[0];
for (int i = 1; i < values.length; i++)
{
  if (values[i] > largest)
  {
    largest = values[i];
  }
}
```

- Note that the loop starts at 1 because we initialize largest with values[0] .
- To compute the smallest element, reverse the comparison.
- These algorithms require that the array contain at least one element.

Element Separators

- When you display the elements of an array, you usually want to separate them, often with commas or vertical lines, like this:

```
32 | 54 | 67.5 | 29 | 35
```

- Note that there is one fewer separator than there are numbers. Print the separator before each element in the sequence except the initial one (with index 0) like this:

```
for (int i = 0; i < values.length; i++)
{
  if (i > 0)
  {
    System.out.print(" | ");
  }
  System.out.print(values[i]);
}
```

Linear Search

- You often need to search for the position of a specific element in an array so that you can replace or remove it.
- Visit all elements until you have found a match or you have come to the end of the array.
- Here we search for the position of the first element in an array that is equal to 100:

```
int searchedValue = 100;
int pos = 0;
boolean found = false;
while (pos < values.length && !found)
{
  if (values[pos] == searchedValue)
  {
    found = true;
  }
  else
  {
```

Removing an Element

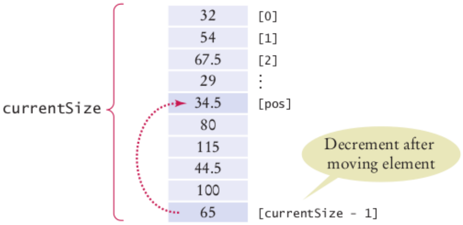
- If the elements in the array are not in any particular order, simply overwrite the element to be removed with the last element of the array, then decrement the currentSize variable.

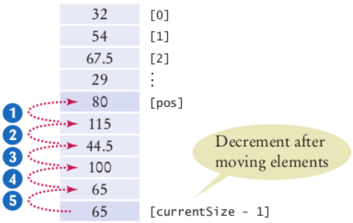
```
values[pos] = values[currentSize - 1];
currentSize--;
```

- The situation is more complex if the order of the elements matters. Then you must move all elements following the element to be removed to a lower index, and then decrement the variable holding the size of the array.

```
for (int i = pos + 1; i < currentSize; i++)
{
  values[i - 1] = values[i];
}
currentSize--;
```

Removing an Element





Inserting an Element

- If the order of the elements does not matter, you can simply insert new elements at the end, incrementing the variable tracking the size.

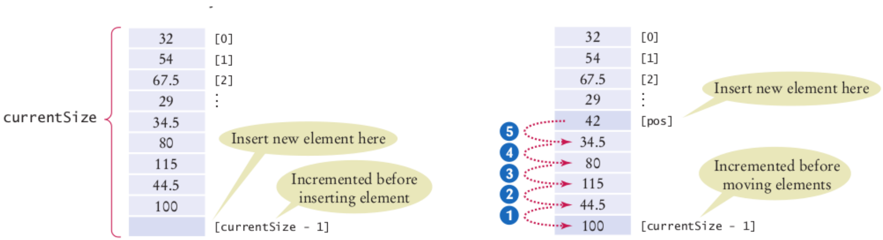
```
if (currentSize < values.length)
{
  currentSize++;
  values[currentSize - 1] = newElement;
}
```

Inserting an Element

- When you insert an element, you start at the end of the array, move that element to a higher index, then move the one before that, and so on until you finally get to the insertion location.

```
if (currentSize < values.length)
{
  currentSize++;
  for (int i = currentSize - 1; i > pos; i--)
  {
    values[i] = values[i - 1];
  }
    values[pos] = newElement;
}
```

Inserting an Element



Swaping an Element

- Consider the task of swapping the elements at positions i and j of an array values . We'd like to set values[i] to values[j].
- But that overwrites the value that is currently stored in values[i] , so we want to save that first:

```
double temp = values[i];
values[i] = values[j];
```

- Now we can set values[j] to the saved value.

```
values[j] = temp;
```