Arrays in Java – Using Arrays

Recall – Length of an Array

Once an array has been created in memory, its size is fixed for the duration of its existence. Every array object has a `length` field whose value can be obtained by appending `.length` to the array identifier.

```
double[] price = new double[100];   // create array with 100 elements
int len = price.length;             // the .length will be 100
```

Since all arrays start numbering at zero, the length is always one greater than the highest index. For example, an array with 100 elements will have a length of 100, but the index will range from 0 to 99.

Traversing Arrays

To access each element in an array, a counted, or `for` loop, is frequently used. Suppose we have an array of 100 `boolean` values, automatically initialized to `false` by default. To change all of these to true, we could write

```
boolean[] flags = new boolean[100];
for (int i = 0; i < flags.length; i++)
{
     flags[i] = true;
}
```

Note that we have used the `flags.length` to determine the upper bound of the loop. As we previously noted, the `.length` field will return a value one greater (in this case, 100) than the maximum index of the array (in this case, 99), which is why our loop must use `i < flags.length` to avoid running past the end of the array. If we attempt to use an index that is outside of the range of the array, Java will throw an *exception* (i.e., report an error).

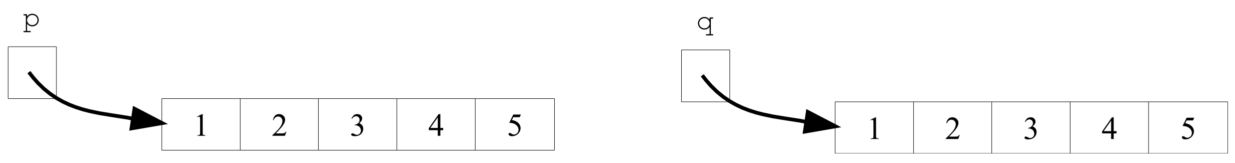
Comparing Arrays

Arrays are objects, and as such, they have the same limitations we experience with other objects. One of those involves the notion of equality.

The array variable is a *reference* to an array object. In other words, it points to a location in memory which holds the array and its data. We must keep this in mind when using the assignment statement (=) and the equality operator (==).

For example, suppose we have made the following declaration:

```
int[] p = {1, 2, 3, 4, 5};
int[] q = {1, 2, 3, 4, 5};
```



Through common sense , it would be reasonable to say they arrays are equal. They contain the same number and type of elements, and each corresponding element contains the same value. Nonetheless, if we were to test their equality,

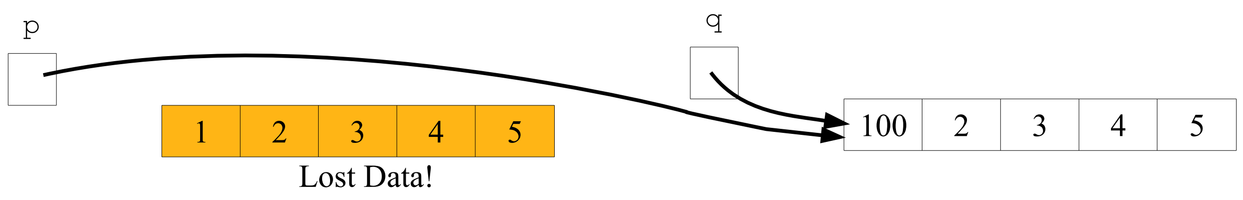
```
System.out.println(p == q);      // false
```

The result would be `false`. The reason is that the variables `p` and `q` are *references* to memory locations, or addresses in memory. Since each array is distinct in memory, it must have a unique address, so they are definitely not equal.

Similarly, using the assignment operator (=) could produce an unexpected situation.

```
p = q;
q[0] = 100;
System.out.println(p[0]);        // outputs 100
System.out.println(p == q);      // now true
```

In this case, the variables p and q now point to the same memory location, and the data contained in the original p array is forever lost.



Arrays as Method Parameters

Like other variables and objects in Java, arrays can be parameters of methods, and can be returned by methods. Since the array variable is a *reference*, the method will make a copy of the reference, so the original cannot be changed by the method. The *referenced **data***, however, can and will be changed by the method unless you keep this in mind (and sometimes you want to change array data using a method, so this is desirable).

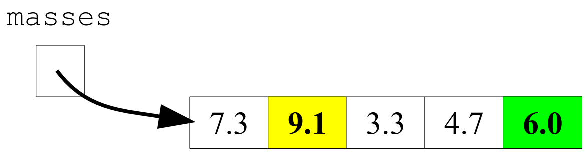
```
public static double[] join (double[] a, double[] b)
{
      double[] result = new double [a.length + b.length];
      int i, j;
      for (i = 0; i < a.length; i++)
      {
            result[i] = a[i];
      }
      for (j = 0; j < b.length; i++, j++)
      {
            result[i] = b[j];
      }
      return result;
}
```

Note that when introducing arrays, we discussed that their size was fixed. While this is true, it is possible to dynamically create a new array of any size, and if necessary, abandon the original array (which, in effect, allows a programmer to dynamically alter the size of an array).

Another example, the method swap shown below will switch the values of two elements in an array of double values. When we tried this using variables, it failed. It succeeds now because an array is a reference to a memory location.

```
public static void swap (double[] list, int i, int j)
{
      double temp = list[i];
      list[i] = list[j];
      list[j] = temp;
}
```

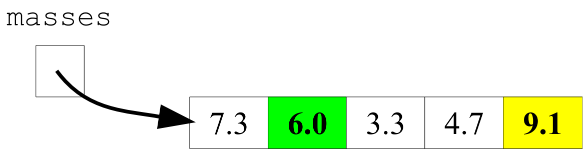
To show the effect of this code, consider the array `masses` shown below:



If we then execute the statement

```
swap(masses, 1, 4);
```

the values of `masses[1]` and `masses[4]` will be switched.



Arrays in Java – Using Arrays

Exercises

1. What would be printed by the following program fragment?

```
int[] list = new int[4];
for (int i = 0; i < list.length; i++)
{
  list[i] = 3 - i;
}
System.out.println(list[1]+2);
System.out.println(list[1+2]);
System.out.println(list[1]+list[2]);
```

2. Suppose that an array sample has been declared as follows:

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

Write one or more statements to perform each task.

a) Initialize all of the elements to one (1).
b) Switch the values at either end of the array.
c) Change any negative values to positive values.
d) Set the variable `sampleSum` to the sum of the values of all the elements.
e) Print the contents of the odd-numbered locations.

3. Write a method `max` that has one double array parameter. The method should return the value of the largest element in the array.

4. Complete the definition of the method `equals` so that it returns true if and only if its two array parameters are identical in every way.

```
public static boolean equals (double[] a, double[] b)
```

5. Write a program that repeatedly prompts the user to supply scores (out of 10) on a test. The program should continue to ask the user for marks until a negative value is supplied. Any values greater than ten should be ignored. Once the program has read all the scores, it should produce a table with the following headings (and automatically fill in the rest of the table):

```
          Score            # of Occurrences
```

The program should then calculate the mean score, rounded to one decimal place.

Solutions

1. What would be printed by the following program fragment?

```
int[] list = new int[4];
for (int i = 0; i < list.length; i++)
{
  list[i] = 3 - i;
}
System.out.println(list[1]+2);
System.out.println(list[1+2]);
System.out.println(list[1]+list[2]);
```

```
4
0
3
```

2. Suppose that an array sample has been declared as follows:

```
int[] sample = new int[SIZE];
```

where `SIZE` is some constant value. Write one or more statements to perform each task.

a) Initialize all of the elements to one (1).

```
for (int i = 0; i < sample.length; i ++)
   sample[i] = 1;
```

b) Switch the values at either end of the array.

```
int temp = sample[0];
sample[0] = sample[sample.length - 1];
sample[sample.length - 1] = temp;
```

c) Change any negative values to positive values.

```
for (int i = 0; i < sample.length; i ++)
   sample[i] = (int)Math.abs(sample[i]);
```

d) Set the variable `sampleSum` to the sum of the values of all the elements.

```
int sampleSum = 0;
for (int i = 0; i < sample.length; i ++)
   sampleSum += sample[i];
```

e) Print the contents of the odd-numbered locations.

```
for (int i = 0; i < sample.length; i ++)
{
   if (i % 2 != 0)
      System.out.println(sample[i]);
}
```

3. Write a method `max` that has one double array parameter. The method should return the value of the largest element in the array.

```
public static double max (double[] array)
{
  double max = array[0];
  for (i = 0; i < array.length; i++)
  {
    if (array[i] > max)
    {
      max = array[i];
    }
  }
  return max;
}
```

4. Complete the definition of the method `equals` so that it returns true if and only if its two array parameters are identical in every way.

```
public static boolean equals (double[] a, double[] b)
{
  boolean equals = (a.length == b.length);
  // passed first test, now check elements
  for (int i = 0; i < a.length; i++)
  {
    // a single failure means total failure
    equals = equals && (a[i] == b[i]);
  }
  return equals;
}
```

5. Write a program that repeatedly prompts the user to supply scores (out of 10) on a test. The program should continue to ask the user for marks until a negative value is supplied. Any values greater than ten should be ignored. Once the program has read all the scores, it should produce a table with the following headings (and automatically fill in the rest of the table):

```
Score           # of Occurrences
```

The program should then calculate the mean score, rounded to one decimal place.

```
class ArraysEx5
{
  public static void main(String [] args)
  {
    // possible scores are 0 through 10, all integer
    int[] countScores = new int[11];
    int score, totalScores = 0, numScores = 0;

    do
    {
      System.out.println("Please enter a mark out of 10 (negative to quit)");
      score = In.getInt();
      if (0 <= score && score <= 10)
      {
        countScores[score]++;   // increment the count of that score
      }
    }
    while (score >= 0);

    // display scores, adding at the same time
    System.out.println("Score    # of Occurrences");
    System.out.println("=====    ================");
    for (int i = 0; i <= 10; i++)
    {
      totalScores += i * countScores[i];
      numScores += countScores[i];
      System.out.println(i + "          " + countScores[i]);
      // The format command below is an alternative to the basic println.
      // You can research this on your own if you wish to experiment.
      //System.out.format("%5d           %d%n", i, countScores[i]);
    }

    System.out.println();
    System.out.println("Average score: " + ((float)totalScores/(float)numScores));
    // The format command below is an alternative to the basic println.
    // You can research this on your own if you wish to experiment.
    //System.out.format("Average score: %.1f%n", ((float)totalScores/(float)numScores));
  }
}
```