Passing an array to a method: Call By Reference

Earlier we saw that when we pass a primitive data type to a static method, the static method parameter gets a copy of the data sent in. We get a different behavior if we pass an object as a parameter rather than a primitive data type. We create objects with the keyword **new**. We haven't defined our own objects yet, but an array is an example of an object we have seen already.

If we change the **contents – e.g. for an array, this would be the data inside the array -** of an object that is passed as a parameter inside a method, then the change is reflected back in the calling code. This is called **call by reference**.   Setting the object itself to something different does not reflect back in the calling code. For example:

```
public class Foo
{
 public static void doesChange(int[] arr)
 {
      arr[0] = 100;
 }

 public static void main(String[] args)
 {
      int[] a = new int[3];
      a[0] = 1;
      doesChange(a);
      System.out.println("Back in main:" + a[0]);     //Prints 100
 }
}
```

First, some mechanics of how to pass an array. **When we define the method, the array parameter is defined using [] after the data type followed by a parameter name**, just like when it is defined in main. We don't need to specify the array size.

When we send an array into a method, we give the variable name but no square brackets. Just the variable name by itself.

The variable is changed back in the caller because we're passing an object as a parameter. Internally, objects are passed to the method by providing the address of the object in memory. This means changes to the object inside the function change the original copy of the data. This is not the case when we pass primitive data types; instead we give a copy of the entire variable instead of providing the address.

Searching an Array

A common operation is to search an array for some value. The simplest way to search an array is through *sequential search*. In sequential search we simply start at the first array element, check to see if it is our target, and if so we stop. If not, we move on to the next element in the array until we reach the end.

As an example, consider an array of student ID's and an array of student names. Just like our trivia game, the index ties the ID with the corresponding name. For simplicity I have just hard-coded the two arrays:

```
        int ids[] = {41, 45, 99, 121, 412, 551};
        String names[] = {"Joe","Carol","Ted","Alice","Bob","Mary"};
```

This says that Joe has ID 41, Carol has ID 45, etc.

Let's say that we want to find the name of the person with ID 121. We can loop through each one of the entries in the ids array until we find a match:

```
        for (int i = 0; i < ids.length; i++)
        {
            if (ids[i]==121)
            {
                System.out.println(names[i]);
                break;
            }
        }
```

We can generalize this a bit better by turning it into a method. The method will take as input parameters the two arrays, the target ID we are searching for, and return "" if no match is found and return the matching name if one is found.

```
    public static String findName(int ids[], String names[], int targetID)
    {
        for (int i = 0; i < ids.length; i++)
        {
            if (ids[i]==targetID)
            {
                return names[i];
            }
        }
        return "";
    }

    public static void main(String[] args) {
        int ids[] = {41, 45, 99, 121, 412, 551};
        String names[] = {"Joe","Carol","Ted","Alice","Bob","Mary"};

        String match;
        match = findName(ids, names, 121);
        System.out.println("Name for ID 121 is " + match);      // Ted
        match = findName(ids, names, 999);
        System.out.println("Name for ID 999 is " + match);      // Blank
    }
```

If we didn't want to print the blank we could check if the return value equals "" and print or do something else.

Note that the id’s are sorted – this allows us to search the list of ID’s more efficiently than sequential search. The book has a section on binary search, which splits the list in half each time. We will hold off on covering binary search in detail until the section on recursion where we will look at a recursive version of binary search.

Sorting – Selection Sort

Sorting an array is useful for many situations. There are hundreds of different sorting algorithms, some more efficient than others. Let’s talk about a relatively simple algorithm named selection sort. The idea behind selection sort is to find the smallest value in the array and copy it to index 0. Then, starting at index 1, find the smallest value in the array and copy it to index 1. Then, starting at index 2, find the smallest value in the array and copy it to index 2, etc.

(example in class)

Let’s try sorting our array of names alphabetically:

```
String names[] = {"Joe","Carol","Ted","Alice","Bob","Mary"};
```

Here is the skeleton for our algorithm. In this case, the length of the array is 6 and the names are at indices 0-5.

```
for (int leftmost = 0;  leftmost < names.length - 1; leftmost++)
{
      Find first alphabetic name in the array from index leftmost to the end
      Swap the first alphabetic name to index leftmost
}
```

How do we find the first alphabetic name? We can use the compareTo method:

```
        System.out.println("A".compareTo("B"));       // Negative
        System.out.println("A".compareTo("A"));       // Zero
        System.out.println("B".compareTo("A"));       // Positive
```

Fleshing this out with more code and putting it into a method:

```
public static void sort(String names[])
 {
     for (int leftmost = 0; leftmost < names.length - 1; leftmost++)
     {
         // Find the first alphabetic name
         int indexOfFirst = leftmost;
         for (int i = leftmost+1; i < names.length; i++)
         {
             if (names[indexOfFirst].compareTo(names[i]) > 0)
                 indexOfFirst = i;
         }
         // names[indexOfFirst] is the first alphabetic name
         // Swap it with names[leftmost]
         String temp = names[leftmost];
         names[leftmost] = names[indexOfFirst];
         names[indexOfFirst] = temp;
     }
 }

 public static void main(String[] args) {
     String names[] = {"Joe","Carol","Ted","Alice","Bob","Mary"};

     sort(names);
     for (int i = 0; i < names.length; i++)
     {
         System.out.println(names[i]);
     }
 }
```