Working With Arrays

Part I

Recall that an array can be declared using curly braces to fill it with values:

```
String[] names={"Adam","Becky","Charlie","Dennis","Ethan","Fred","Gary","Heath"};
```

Listing its type, indexes, and values can summarize the above array:

Before

`String[] names`

index	Value
0	"Adam"
1	"Becky"
2	"Charlie"
3	"Dennis"
4	"Ethan"
5	"Fred"
6	"Gary"
7	"Heath"

Once an array has been created, it is often more useful to use methods to work with it. As with any method, the array can be "passed" to it as a parameter so long as the type matches:

```
public static void doSomething(String[] arr) {
     arr[0]="Aaron";
}
```

It is important to realize two things when working with an array that has been passed to a method as a parameter. First, even though the name of the array is different when working inside the method, for all intents and purposes, you are still working with, and therefore can alter the original array!

For example, calling the `doSomething(names)` method would have the effect of changing `names[0]` to have the value `"Aaron"` because the `names` array was used as the parameter so the arrays `arr` and `names` are the same thing within the method. As a result, after calling the method, the new array summary of `names` would look like this:

After

`String[] names`

index	Value
0	"Aaron"
1	"Becky"
2	"Charlie"
3	"Dennis"
4	"Ethan"
5	"Fred"
6	"Gary"
7	"Heath"

Be careful not to change an array value within a method unless the goal is to actually change the value in the original array!

It is usually more efficient to create methods to work with arrays because then the same method can be used on more than one array. Common operations that can take place in a method include:

- Outputting the contents of an array to the console formatted in a particular way
- Getting the contents of an array in the form of a String
- Adding an element to an array
- Removing an element from an array
- Searching for a value in the array
- Searching an array for a maximum or minimum value
- Moving elements within the array, the most common operation involves "swapping" two elements
- Merging two arrays

Many of these operations use a common algorithm to repeat whatever the general purpose is of the method on each element of an array.

Usually a loop will be used to iterate over each value stored in the array. In most cases a "for loop" is the easiest way to accomplish this because the start and end values are known (or can be calculated) and each iteration of the loop increases the iterator by one.

The typical construction starts with the first element of the array, which is 0, and ends with the last element, which can be calculated using the array name and the length property.

```
for (int i=0; i<array.length; i++)
```

Outputting a Formatted Array

For example, outputting the contents of an array formatted conveniently:

```
public static void printArray(String name, String[] arr) {
  for (int i=0; i<arr.length; i++) {
    System.out.println(name+"["+i+"]="+arr[i]);
  }
}
```

Calling this method with the names array, and passing “names” as the second parameter after the names array was initialized:
`printArray("names", names);`

Generates this console output:

```
names[0]=Adam
names[1]=Becky
names[2]=Charlie
names[3]=Dennis
names[4]=Ethan
names[5]=Fred
names[6]=Gary
names[7]=Heath
```

Turning an Array into a String

The problem with directly outputting an array’s contents to the console as in the above example is that the programmer has no way to modify the output to meet their needs. Another way to process an array for eventual output is to turn the contents into a String.

This process is relatively simple once the desired format is chosen because it is remarkably similar to simply outputting the contents to the console. But instead each value that would be output to the console is stored in an accumulator String. Then the accumulator is returned.

Using the same `String` array called `names`:
`String[] names={"Adam","Becky","Charlie","Dennis","Ethan","Fred","Gary","Heath"};`

For this example, the desire is for the String to take the form of a short-cut array initialization, so curly braces surround the contents of the array and each element of the array has quotes around it and commas separate them.

The pseudo-code solution would look like this:

1. Initialize an accumulator String so it has an open curly brace.
2. If the array is null, simply return the String literal `"null"`.
3. Iterate over each index in the array.
4. Add the array contents to the accumulator for that index surrounded by quotes.
5. If the array index is less than the array length minus 1 then add a “,” to the accumulator.
6. Add a closed curly brace the end of the accumulator.
7. Return the accumulator.

Below is a summary of the information about the method to be written:

Method Attribute	Description	JAVA
Return Type	A `String`.	`String[]`
Method Name	This method will convert a String array to a String.	`toString`
Parameter(s)	The array to convert.	`String[] arr`
Return value	A String in the form {"e_0","e_1",…,"e_l"} or "null".	`String`

Add an element to an Array

Arrays have a fixed size when they are created so it is no simple task to add an element after the array's size has been set in the declaration!

The process can best be described as creating a new array that has room for the new element and then copying the contents of the original array to the new array with the added element placed at the end.

Now examine another `String` array called `names`:
`String[] names={"Aaron","Brandy","David","Edith"};`

Next assume that the value `"Charlie"` needs to be added to the end of the array.

The pseudo-code solution would look like this:

1. Create a new array with a size of 5.
2. Copy the elements from index 0 to 3 into the new array.
3. Assign `"Charlie"` to the new array at index 4.

Next the pseudo-code should be made general so that it can be turned into a method where any value can be added to the end of the array.
Below is a summary of the information about the method to be written:

Before
`String[] names`

index	Value
0	"Aaron"
1	"Brandy"
2	"David"
3	"Edith"

After
`String[] names`

index	Value
0	"Aaron"
1	"Brandy"
2	"David"
3	"Edith"
4	"Charlie"

Method Attribute	Description	JAVA
Return Type	A `String` array with the new element added.	`String[]`
Method Name	This method will add an element to an array.	`add`
Parameter(s)	The `String` element to add to the array.	`String s`
	The array to add the element to.	`String[] arr`
Return value	A new array with one more element in it.	`String[]`

Here is the complete pseudo-code:

1. Get the value to be inserted and the array as parameters.
2. If the array is null create a new array with a size of 1 and give the first element the value of `s`.
3. Create a new array that has a size equal to the size of the original array plus one.
4. Use a loop to assign elements from the original array to the new array using the same indexes.
5. Assign the new element to the new array at the last index of the new array.
6. Return the new array since it was created as a local variable within a method.

The method signature would read:
`String[] add(String s, String[] arr)`

Assume an array: `letters={"A","B"};`

And a call to the method `letters=add(letters, "C");`:

Use Chart B to follow this process step by step for the sample array and element:

Step 1:
`arr={"A","B","C","E","F","G"};`
`s="C";`
Step 2 is skipped because `arr` is not `null`.
Step 3: Create a new array, `nArr` with a length 1 larger than `arr`.
Step 4: Copy the elements to the new array.
Step 5: Add the new element `s` to the new array.
Step 6: Return the new array.

Chart B: Adding an Element

After Step 3

arr		nArr	
index	Value	index	Value
0	"A"	0	null
1	"B"	1	null
		2	null

After Step 4

arr		nArr	
index	Value	index	Value
0	"A"	0	"A"
1	"B"	1	"B"
		2	null

After Step 5

arr		nArr	
index	Value	index	Value
0	"A"	0	"A"
1	"B"	1	"B"
		6	"C"

Add an element to an Array at an index

This version of adding an element is much like adding an element to the end of an array, but the new element can be put anywhere in the existing array instead.

The process can best be described as creating a new array that has room for the new element and then copying the contents of the original array to the new array with the added element inserted.

Examine the `String` array called `names` again:
`String[] names={"Aaron","Brandy","David","Edith"};`

Next assume that the value `"Charlie"` needs to be added to the array so that it has an index of 2, moving the other elements after it down to make room.

The pseudo-code solution would look like this:

1. Create a new array with a size of 5.
2. Copy the elements from index 0 to 1 into the new array.
3. Assign `"Charlie"` to the new array at index 2.
4. Copy the elements from index 2 on into the new array starting at index 3.
5. Assign the new array to the old array.

Before
`String[] names`

index	Value
0	"Aaron"
1	"Brandy"
2	"David"
3	"Edith"

After
`String[] names`

index	Value
0	"Aaron"
1	"Brandy"
2	"Charlie"
3	"David"
4	"Edith"

Next the pseudo-code should be made general so that it can be turned into a method where any value can be inserted at any valid index position.
Below is a summary of the information about the method to be written:

Method Attribute	Description	JAVA
Return Type	A `String` array with the new element inserted.	`String[]`
Method Name	This method will add an element to an array.	`add`
Parameter(s)	The `String` element to add to the array.	`String s`
	Index where the element should be added.	`int index`
	The array to add the element to.	`String[] arr`
Return value	A new array with one more element in it.	`String[]`

Here is the complete pseudo-code:

1. Get the value to be inserted, the index where it should be placed, and the array as parameters.
2. If the index isn't within the bounds of the array, make it equal to the size of the original array thus it will be safely added as the last element of the array.
3. Create a new array that has a size equal to the size of the original array plus one.
4. Use a loop to assign elements from the original array to the new array using the same indexes until the new element's index is reached.
5. Assign the new element to the new array at the new element's index.
6. Use another loop to continue copying elements from the original array to the new array, putting the elements in the new array at an index equal to the original array index plus one.
7. Return the new array since it was created as a local variable within a method.

The method signature would read:
`String[] add(String s, int index, String[] arr)`

Assume an array: `letters={"A","B","C","E","F","G"};`

And a call to the method `letters=add("D", 3, letters)`:

Use Chart C to follow this process step by step for the sample array, element, and insertion index:

Step 1: Get the array value

```
arr={"A","B","C","E","F","G"};
s="D";
index=3;
```

Step 2: Is `index` within the bounds of `arr`?

```
index>0 && index<arr.length;
```

Step 3: Create the new array

```
String[] nArray=new String[arr.length+1];
```

Step 4: Copy to the new array up to the insertion index

```
for (int i=0; i<index; i++) {
  nArray[i]=arr[i];
}
```

Step 5: Add the new element at the index

```
nArray[index]=element;
```

Step 6: Continue copying the remaining elements

```
for (int i=index+1; i<nArray.length; i++) {
  nArray[i]=arr[i-1];
}
```

Step 8: Return the new array

```
return nArray;
```

Chart C: Adding Element at Index

After Step 3

arr		nArray	
index	Value	index	Value
0	"A"	0	null
1	"B"	1	null
2	"C"	2	null
3	"E"	3	null
4	"F"	4	null
5	"G"	5	null
		6	null

After Step 4

arr		nArray	
index	Value	index	Value
0	"A"	0	"A"
1	"B"	1	"B"
2	"C"	2	"C"
3	"E"	3	null
4	"F"	4	null
5	"G"	5	null
		6	null

After Step 5

arr		nArray	
index	Value	index	Value
0	"A"	0	"A"
1	"B"	1	"B"
2	"C"	2	"C"
3	"E"	3	"D"
4	"F"	4	null
5	"G"	5	null
		6	null

After Step 6

arr		nArray	
index	Value	index	Value
0	"A"	0	"A"
1	"B"	1	"B"
2	"C"	2	"C"
3	"E"	3	"D"
4	"F"	4	"E"
5	"G"	5	"F"
		6	"G"

Remove an element from an Array

Removing an element follows the same basic algorithm for adding one except that the new array is smaller rather than larger!

The process can best be described as creating a new array that has room for one less element than the original array and then copying the contents of the original array to the new array skipping the element to be removed along the way.

The method signature is very similar to adding an element, only with one less parameter.

Below is a summary of the information about the method to be written:

Method Attribute	Description	JAVA
Return Type	A `String` array with the new element inserted.	`String[]`
Method Name	This method will add an element to an array.	`remove`
Parameter(s)	Index where the element should be added.	`int index`
	The array to add the element to.	`String[] arr`
Return value	A new array with one more element in it.	`String[]`

Here is the complete pseudo-code:

1. Get the array and the index of the element to be removed as method parameters.
2. If the index is outside the bounds of the array simply copy the existing array and return it.
3. Create a new array that has a size equal to the size of the original array minus one.
4. Use a loop to assign elements from the original array to the new array using the same indexes until the element to be removed is reached.
5. Use another loop to continue copying elements from the original array to the new array, putting the elements in the new array at an index equal to the original array index minus one.
6. Return the new array.

The method signature would read: `String[] remove(int index, String[] arr)`

Given an array: `letters={"A","B","C","D","E","F","G"};` and a call to the method `letters=remove(letters, 3)`

Use Chart C to follow this process step by step:

Step 1: Get parameter values

```
String[] arr={"A","B","C","D","E","F","G"};
int index=3;
```

Steps 2: Is `index` within the bounds of `arr`?

```
index>0 && index<arr.length;
```

Step 3: Create the new array

```
String[] nArray=new String[arr.length-1];
```

Step 4: Copy elements to the new array up to the index

```
for (int i=0; i<index; i++) {
  nArray[i]=arr[i];
}
```

Step 5: Copy elements after the index to the new array

```
for (int i=index+1; i<nArray.length; i++) {
  nArray[i]=arr[i-1];
}
```

Step 7: Return the new array

```
return nArray;
```

Chart C: Removing an Element

After Step 3

arr		nArray	
index	Value	index	Value
0	"A"	0	null
1	"B"	1	null
2	"C"	2	null
3	"D"	3	null
4	"E"	4	null
5	"F"	5	null
6	"G"		

After Step 4

arr		nArray	
index	Value	index	Value
0	"A"	0	"A"
1	"B"	1	"B"
2	"C"	2	"C"
3	"D"	3	null
4	"E"	4	null
5	"F"	5	null
6	"G"		

After Step 5

arr		nArray	
index	Value	index	Value
0	"A"	0	"A"
1	"B"	1	"B"
2	"C"	2	"C"
3	"D"	3	"E"
4	"E"	4	"F"
5	"F"	5	"G"
6	"G"		