Arrays

An array is a collection of variables of the same type, stored in a contiguous chunk of memory.

We can create an array to hold variables of any type:

int[] array1 - array1 is an array that will store ints

JFrame[] array2 - array2 is an array that will store JFrames

int[][] array3 - array4 is an array that will store int[]. That is, it is an array that stores arrays that store ints.

We can initialize the array using the new operator and stating how many variables (elements) will be in the array:

array1 = new int[100];

array1 now stores the address for an array that contains 100 int variables.

array2 = new JFrame[30];

array2 now stores the address for an array that contains 30 JFrame variables.

Now, to access each element, we again use the square brackets. To store a value in the first element (variable) of the array:

array1[0] = 12;

Remember, the we start counting from 0 in Java!

Each element of an array is a variable of the given type. So all the rules of variables apply:

array1[2] = 3.15 <- Illegal! We are trying to store a double in a variable of type int

array1[2] = 7 <- Legal!

Array disadvantage:

Because the array is stored in contiguous memory, we can not change the length (number of variables) of the array once it is created.

Array advantage:

Because the array is stored in contiguous memory, we can access each element in a single step, no matter how large:

int[] a = new int[1000000];

For example, to access a[95436], we do not need to run through the array to the 95436'th entry.

We know the address of a (it is stored in the variable a), we know the byte size of the type of the array (for example, each int is 4 bytes)

and we know which one we want (index 95436).

So the address of a[95436] is: (address of a) + 4 \* 95436

Initializing an array.

We need to keep track of our rules of variables and be sure to always declare and initialize our arrays.

(In Java, arrays get default values of 0, in C they get no default values.)

int[] a; <= a is declared as the address of an array, but it is not initialized

a = new int[10] <= now a is initialized to store the location of an array of int. However, each of the 10 entries in the array are uninitialized variables.

Here is one way to initialize. In this case, we will create an array of JFrame and initialize each to a new window:

JFrame[] frames = new JFrame[10];

for (int i = 0; i < length; i = i + 1) {

frames[i] = new JFrame();

}

Our first method with arrays:

Create a method that takes an int array and an int. The array is full, but we need to put the int value at the end of the array.

We can't change the size of an array, so instead, we must create a brand new array, copy the values over, and then put the new value at the end.

public int[] append(int[] array, int x) {

int[] array2 = new int[array.length + 1];

for (int i = 0; i < array.length; i = i + 1)

array2[i] = array[i];

array2[array2.length - 1] = x;

return array2;

}

First, note that we needed a loop to do the copy. If we write "array2 = array;" we instead copy the address of array to array2.

That has the affect of having both array2 and array point to the same array.

Second, note that we copied x in outside the loop. We could have tried to also include x inside the loop, but that would mean an if statement

and more complicated code. To keep your code simple, remember to have each loop accomplish one task.

Third, note that we had to return array2. If we did not, we would lose the array2.

array2 is a local variable and so it is allocated from the stack.

When the method ends, all stack variables are wiped out. If we lose array2, we lose the address to the new array in the heap.

By returning the new address, we can have it outside the method.

In class it was suggested we have an assignment statement: array = array2. This would copy the address of array2 to array, but it has the same problem

because array is also allocated from the heap. Thus array will be lost once the method ends.

Example:

myArray = append(myArray, 11);

Will create a new array one larger than myArray, copy the data over, add 11 to the end, and then return the address of the new array, and that address is stored in myArray.

myArray now points to a new array one larger than the array it originally pointed to. The old array is still in the heap. Java will eventually de-allocate it if there are no other variables storing its address.

Example 2: Reversing the contents of an array

(Note that, unlike with String, we can change the contents of an array, and so the reverse can happen in-place.

public void reverse(int[] array) {

}

Why void? Think about how reverse will be called:

int[] a = new int[....];

.... code to add date to the array

reverse(a);

The array is allocated in the heap. a stores the address of the array.

When we call reverse, the address of a is copied to the the variable array. So, while inside the reverse method, both array and a point to the same heap memory.

As a result, when we exit, all changes made to array inside the method will also be seen by a outside the method.

(Note the example of last class required us to return the array because the -address- of the array was changed by the method, not just the array's contents.)

How do we reverse an array?

We step through from the front to ???

Each every element we see, we swap it with the corresponding one at the end of the array.

Now, we can see that we stop at the middle of the array. (Trace what would happen if we kept going all the way to the end of the array!)

public void reverse(int[] array) {

// loop subgoal: after i iterations, the first i and last i elements of array have changed places

for (int i = 0; i < array.length / 2; i = i + 1) {

}

}

How do we swap values between array entries (i.e. how do we swap values between two variables?)

If we copy from one variable to a second variable, we will lose the data in the second variable unless we remember it first!

There is only one way to remember a value in Java!

public void reverse(int[] array) {

// loop subgoal: after i iterations, the first i and last i elements of array have changed places

for (int i = 0; i < array.length / 2; i = i + 1) {

int save = array[array.length - 1 - i];

array[array.length - 1 - i] = array[i];

array[i] = save;

}

}