CMSC131

Data Structure: an array which we control

If ArrayList had much of what you wanted but not all, would you want to…

33% A. be able to edit ArrayList.java to add what you wanted

33% B. copy ArrayList.java into a different file, rename it, and edit that

33% C. be able to use the existing ArrayList as a tool while make a new class

Response Counter

The array: The Concept

There are a wide variety of data structures that we can use or create to attempt to hold data in useful, organized, efficient ways.

Similar to an **`ArrayList`**, an **array** is a linear, contiguous, homogenous structure that can hold an arbitrary number of elements but is available in all languages.

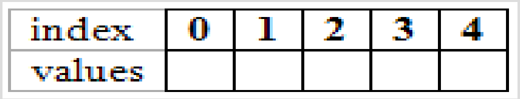
However, for this structure we need to know how many we want it to hold in advance. Once we allocate the array, its size can not be altered.

The array: Some Properties

Elements in an array of size ***n*** are indexed (some languages do 1..***n***, many like Java use 0..***n***-1) and that is the only way we can access items inside.

The data structure has a "first" and "last" position.

In an array with more than 1 element, any position other than the first has a "previous" position and any position other than the last has a "next" position.



index	0	1	2	3	4
values					

The array in Java

In Java, an array is an object and we will have a variable serve as a reference to that object.

We will indicate the type of elements that will be stored when we declare it.

```
datatype[] arrayName;
```

We will allocate the actual array.

```
arrayName = new datatype[size];
```

We will access a position within an array by using the reference variable with the position inside brackets.

```
arrayName[position]= value;
```

Our Simple Problem

Since we've explored the idea of lists already, let's jump to the same example problem that got us started there; what if you had to write a program that would allow the user to specify how many numbers they wanted to enter, then read them in, and print them out backwards?

What might we need to do?

Some of the steps are the same as when using an **`ArrayList`**, but some are "new" to us:

- Declare some variables like an integer to store the answer when we ask the user how many numbers they want to enter and a variable to act as a reference to an array.
- Get the size request from the user.
- Allocate an array big enough to hold them all.
- Allow the user to enter those numbers one at a time and store them in the array.
- Traverse through the array backwards and print them out.

What might we need to do?

```
int[] dataArray;
int numberOfElements;
```

- Get the size request from the user.
- Allocate an array big enough to hold them all.
- Allow the user to enter those numbers one at a time and store them in the array.
- Traverse through the array backwards and print them out.

What might we need to do?

```
int[] dataArray;
int numberOfElements;
numberOfElements = sc.nextInt();
```

- Allocate an array big enough to hold them all.
- Allow the user to enter those numbers one at a time and store them in the array.
- Traverse through the array backwards and print them out.

What might we need to do?

```
int[] dataArray;
int numberOfElements;
numberOfElements = sc.nextInt();
dataArray = new int[numberOfElements];
```

- Allow the user to enter those numbers one at a time and store them in the array.
- Traverse through the array backwards and print them out.

What might we need to do?

```
int[] dataArray;
int numberOfElements;
numberOfElements = sc.nextInt();
dataArray = new int[numberOfElements];

for (int i=0; i<numberOfElements; i++) {
     dataArray[i] = sc.nextInt();
} //I am using “i” to make this fit page
```

– Traverse through the array backwards and print them out.

What might we need to do?

```
int[] dataArray;
int numberOfElements;
numberOfElements = sc.nextInt();
dataArray = new int[numberOfElements];

for (int i=0; i<numberOfElements; i++) {
     dataArray[i] = sc.nextInt();
} //I am using "i" to make this fit page

for (int i=numberOfElements-1; i>=0; i--){
     System.out.print(dataArray[i] + " ");
} //I am using "i" to make this fit page
```

Length of an Array

After allocating an array, we could have a variable that contains the size we requested.

In some languages (including Java) an array will have an *instance variable* within the object that stores the size, but some other languages (like C++) do not, so we ***need*** to keep track of it.

- In Java, if you can access the size of the array via the instance variable **`length`**. **`arrayName.length`**
- The **`.length`** field is a read-only value, so you cannot grow an array by changing this number!

Other differences from `ArrayList`

An array can hold either primitives data directly or hold references to objects.

When an array is created, each position of the array is initialized with a default value.

- If it is an array of object references, they are all initialized to null.
- If it is an array of primitive values, numeric ones are initialized to 0, boolean is initialized to false, and char is initialized to the null character.

Some operations on an Array

You can allocate and initialize an array at the same time if you know all of the information in advance.

```
int[] firstFivePrimes = {2,3,5,7,11};
```

Remember that the variable we are using is a reference to an object. The following just makes a copy of the reference.

```
int[] notCopied = firstFivePrimes;
```

We'll look at three types of array copies a little later in our discussion.

"Growing" an Array

We can't really grow an array once it has been allocated, but what we ***can*** do is allocate a new array, copy the information from the old one into the new one, and move the array reference to point to this new array object.

Imagine we have an array named **`myData`** and we want to make it twice as large without losing existing data…

```
int[] tempName = new int[myData.length*2];
for (int i=0; i<myData.length; i++){
  tempName[i] = myData[i];
} //I am using "i" to make this fit page
myData = tempName;
```

Note that this is the type of thing done behind the scenes when an **`ArrayList`** needs to grow.

Array of Objects

Array elements can be references to objects as well. If so, the array will hold the references to those objects, not the actual objects.

```
String[] topTwo = {"EpVII", "Avatar"};

StringBuffer[] topTwo = new StringBuffer[2];
     topTwo[0] = new StringBuffer("EpVII");
     topTwo[1] = new StringBuffer("Avatar");

String[] topTen = new String[10];
//fill them in…
```

Reference, Shallow, Deep Copies

Reference Copy

```
Student[] array1 = new Student[10];
//some code here which fills in the array with data
Student[] array2 = array1;
```

Shallow Copy

```
Student[] array1 = new Student[10];
//some code here which fills in the array with data
Student[] array2 = new Student[array1.length];
for (int i=0; i<array1.length; i++) {
   array2[i] = array1[i];
} //I am using "i" to make this fit page
```

Reference, Shallow, Deep Copies

Deep Copy

```
Student[] array1 = new Student[10];
//some code here to fill in the array with data
Student[] array2 = new Student[array1.length];
for (int i=0; i<array1.length; i++) {
   array2[i] = new Student(array1[i]);
} //I am using "i" to make this fit page width
```

What is the danger in the above approach that neither reference nor shallow copies faced?

`Arrays` class

There is a useful library class **`Arrays`** which contains a variety of static methods.

One subset of these that can be useful when exploring arrays is the group of **`toString()`** methods which take an array and generates an ASCII visualization of that array.

We can use this to easily see the contents of an array.

```
System.out.println(Arrays.toString(array1));
```

An array as an argument

A reference to an array can be passed as an argument into a method.

You do NOT specify the size of the array since the array itself isn't really being passed into the method, just the reference to it.

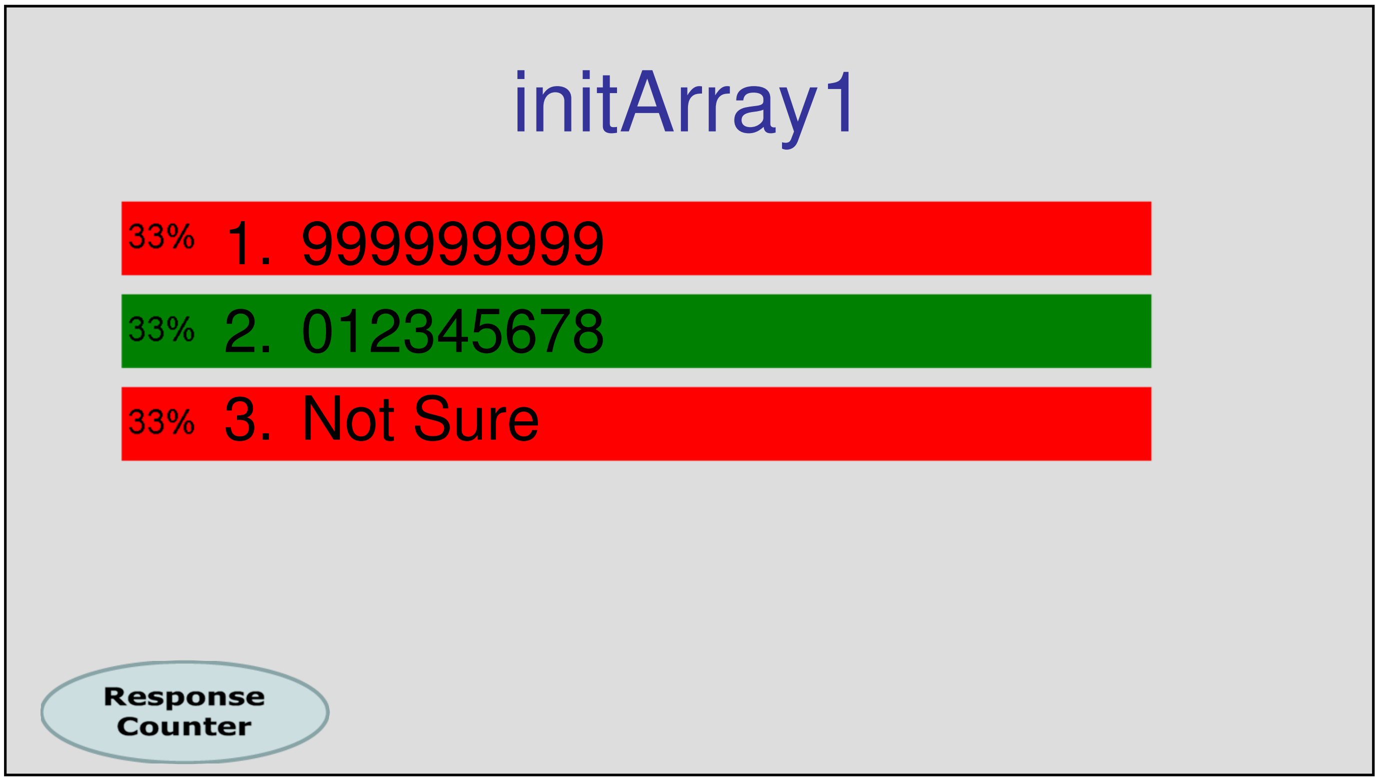
Once the reference to the array is passed into a method, that method can access and alter the elements stored within the array.

Let's look at **`ArrayParameter.java`** and **`ArrayParameterDriver.java`**

Let's look at some code…

http://www.cs.umd.edu/class/fall2019/cmsc131-010X/Notes/Code/ArrayParameter.java

http://www.cs.umd.edu/class/fall2019/cmsc131-010X/Notes/Code/ArrayParameterDriver.java



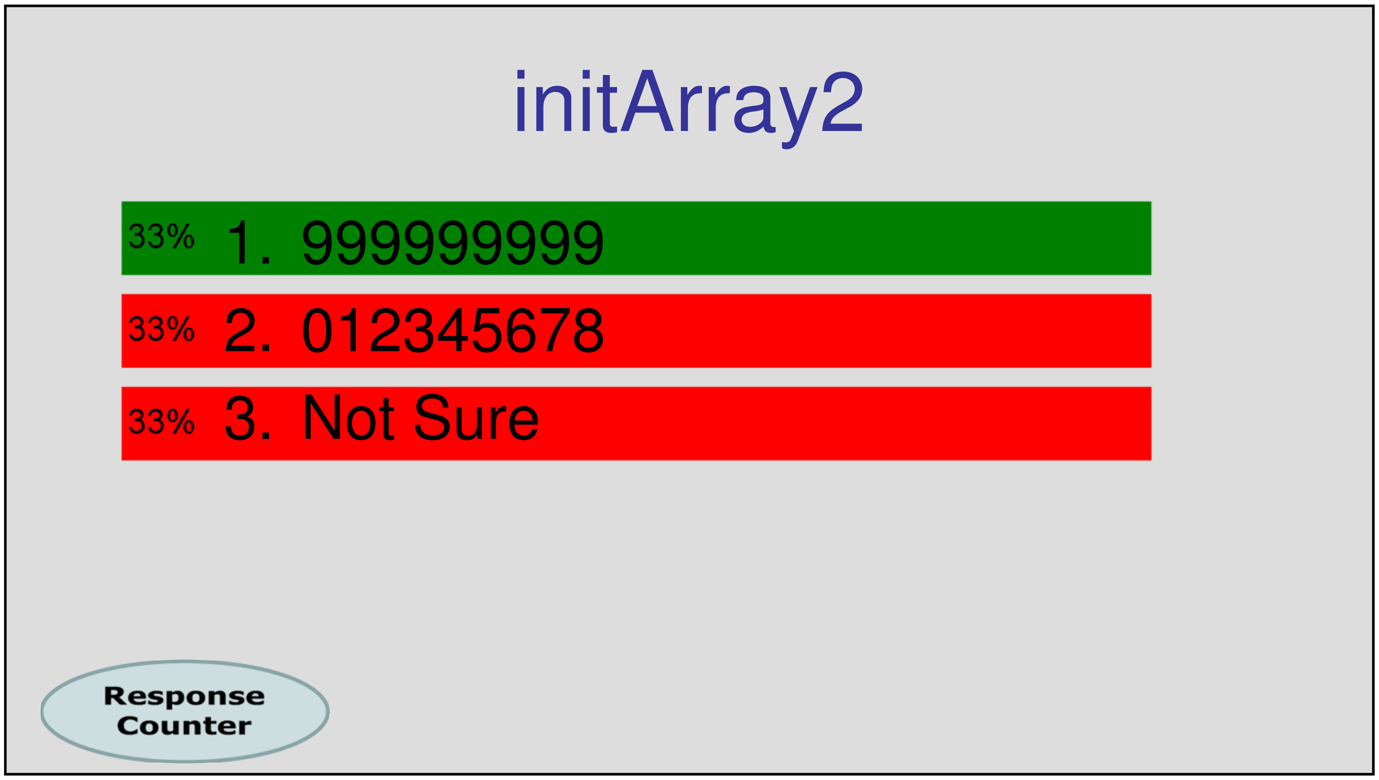
initArray1

33% 1. 999999999

33% 2. 012345678

33% 3. Not Sure

Response Counter



initArray2

33% 1. 999999999

33% 2. 012345678

33% 3. Not Sure

Response Counter

"Privacy" Issues

We've discussed some of the issues of data privacy in an object-oriented language like Java.

With arrays, even though the size of the array is immutable, the ***contents*** aren't, so even immutable objects are "tricky" since you could replace the object itself in an array once you had a reference to the array even if the array reference itself was **`final`**.

– There is ***no way*** to make the contents of an array immutable in Java.