

Lecture 3

Arrays and Linear Search

Last time:

We had a very short overview of the Collections Framework in Java.

Conceptually, collections are objects that group other objects.

In Java, collections are a set of interfaces and classes that are available for you to use in the `java.util` package.

There are three major interfaces: List, Set and Map.

These interfaces are implemented as ArrayList, LinkedList, HashSet, TreeSet, HashMap and TreeMap.

We also saw that there are major operations we can perform on them:

- Addition (Insertion)
- Removal (Deletion)
- Sorting
- Searching
- Iteration (Traversal)
- Copying (Cloning)

Now we will kick off our explorations of data structures, starting with Arrays.

Problem at hand

I would like to have my class roster of 08-722. And, I would like to develop a very simple application for that.

For the application, what kind of data should I use?

How about student id numbers? (No duplicates)

And what kind of operations do I need?

Conceptual View of Arrays

Given the following array with student ids!

0	1	2	3	4	5	6	7	8	9	10
84	61	15	73	26	38	11	49	53	32	

Addition (Insertion)

Add a new id, 91, at the end!

0	1	2	3	4	5	6	7	8	9	10
84	61	15	73	26	38	11	49	53	32	91

Add a new id, 23, at the beginning into the initial array

0	1	2	3	4	5	6	7	8	9	10
23	84	61	15	73	26	38	11	49	53	32

Searching

To search for 84, how many items should we go through? _____

How about searching for 10? How many items should we go through?

This is linear search. Also called sequential search, it is the simplest search algorithm but its worst-case cost is proportional to the number of elements.

Removal (Deletion)

After removal of 73 (Cannot have a hole!)

0	1	2	3	4	5	6	7	8	9
84	61	15		26	38	11	49	53	32

0	1	2	3	4	5	6	7	8	
84	61	15	26	38	11	49	53	32	

Arrays in Memory

Typically, array elements are stored in adjacent memory cells. And index is used to calculate an offset to find the desired element.

Primitive int in Java are 4 bytes (32 bits).

Address	200	204	208	212	216
Content					

If we want the third data value, we take the address of the start of the array and the offset * the size of an array element to find the element we want.

Location of data[2] is $200 + 2 * 4 = 208$

Arrays in Java

Arrays in Java can hold primitives and references. Good for storing and accessing a sequence of data.

An array in Java has *one public method*, `clone()`, and only *one immutable field*, `length`.

To ease some frustrations you might have, Java has the `java.util.Arrays` class. The `java.util.Arrays` class has some static methods that are useful working with arrays.

Examples of using static Arrays methods

```
int[] a = {1,2,3,4,5};  
int[] b = {1,2,3,4,5};
```

```
if(Arrays.equals(a,b))  
    System.out.println("Arrays with identical contents");
```

How about this? Which value will be printed?

```
if(a == b)  
    System.out.println("a == b is true");  
else  
    System.out.println("a == b is false");
```

Printing arrays? Which one will print the values of the array `c`?

```
int[] c = {3,4,2,5,1}  
System.out.println(c);  
System.out.println(Arrays.toString(c));
```

How about sorting?

```
Arrays.____(c);
```

How about copying?

```
int[] d = Arrays.copyOf(c, c.length);
```

The second parameter specifies the length of the new array. It could be less, equal or bigger than the original length.

However, there is a better or flexible way to copy data between arrays which is done by using `System.arraycopy()` method.

It requires five arguments. Here is its signature.

```
public static void arraycopy(Object source, int srcIndex, Object  
destination, int destIndex, int length)
```

What does this code print?

```
int[] e = {1,2,3,4,5};  
int[] f = new int[e.length];  
System.arraycopy(e, 0, f, 0, 3);  
System.out.println(Arrays.toString(f));
```

The other choice for copying is using cloning.

```
int[] g = {1,2,3,4,5};  
int[] h = g.clone();
```

In cloning process, it creates a new array of the same size and type and copies (**shallow copy**!) the elements from the original array into the new array.

Insertion and Deletion

Once again, arrays in Java have only one **immutable** field length. It is useful for you to find the length or size of an array.

```
int arrayLength = h.length;
```

However, once an array is created, its length is fixed and cannot be changed.

Now, how can we resize the array?

It is done by creating a new array with a bigger size and copying all of the values from the old array to the new array.

For example, the following code is to delete an element from an array of char data set.

```
// removes an item at the specified position and returns the new array
public char[] delete(char[] data, int pos) {
    if(pos >= 0 && pos < data.length) {
        char[] tmp = new char[_____];
        System.arraycopy(data, 0, tmp, 0, _____);
        System.arraycopy(data, _____, tmp, pos, _____);
        return tmp;
    } else {
        return data;
    }
}
```

Efficiency of Arrays (Worst-case in Big-O notation)

Insertion at back: _____
 Insertion at front: _____
 Insertion in middle: _____
 Searching (using linear search): _____
 Deletion: _____
 Access to an element with its index: _____

Summary and Review

Arrays might be a good choice if:

- The amount of data is reasonably small
- The amount of data is predictable in advance

Can you see why now?

Remember! Arrays have only one immutable field, length.

In lecture 4, we will look into ArrayList to find out how it handles the issue of the fixed size of arrays and its capabilities.