Data Structures

- ways of storing and controlling data.

Arrays

```
// ArrayType[] var_name = new ArrayType[size];
// Static size

int[] numbers = new int[8];
String[] s1 = {"hi", "bye", "good night"};
String[] s2 = s1;
// NOT copied. referenced so s1 and s2 are the same object array.

numbers[0] = 100;
numbers[7] = 90;

for(int n : numbers) {

}
```

Methods	Return
`array.length`	int
`array[index]`	arrayType

> Import java.util.Arrays

Methods	Return
`Arrays.binarySearch(array,value)`	
`Arrays.copyOf(array, length)`	
`Arrays.equals(array1, array2)`	boolean (same order)
`Array.fill(array, value)`	array
`Arrays.sort(array)`	array
`Arrays.toString(array)`	string

ArrayList

- mistakes to check: removing index (it changes every statement) and removing object
- must confirm if you did remove with if statement.

```
String UserID2Remove = "K";

if(AL.remove(UserID2Remove)) {
    System.out.printf("%s Removed", UserID2Remove);
} else {
    System.out.printf("%s not found", UserID2Remove);
}
```

```
import java.util.ArrayList;

// u can have your own custom type
ArrayList<Integer> numbers = new ArrayList<>(1, 2, 3);
numbers.add(1);
numbers.add(2);
numbers.add(3);


numbers.remove(0); // removes index not value "0" ADDITIONALLY SHIFTS EVERYTHING
TO THE LEFT CHANGING ALL VALUES INDEX
numbers.remove(1); // [2, 3] REMOVING INDEX 1 MAKES IT [2]

System.out.println(numbers); // [2]
numbers.add(1);
numbers.size(); // 2 since we added another element
numbers.get(0); // index 0 is 1;
```

Methods	Return
`myArrayList.add(value)`	
`myArrayList.add(index, value)`	pushes everything in ahead forward
`myArrayList.clear()`	void
`myArrayList.indexOf(value)`	int index
`myArrayList.size()`	int
`myArrayList.sort()`	void
`myArrayList.remove(int index)`	`boolean`
`myArrayList.remove(Object o)`	`boolean`removes first index that there is an object. does not remove duplicates

LinkedList

HashSet

- Data structure set that does not allow duplication.
- There is no index and no specific order.
- It is a set and it hashs the data with a "hash function/table" that allows the computer to find the memory address (look up table) to the values.
- HASHSETS ARE REFERENCES OH NOES

```
HashSet<String> hs = new HashSet<String>();

hs.add("bobby");
hs.add("google");
hs.add("samsung");

System.out.println(hs);

for(String s : hs){
    System.out.print(s);
}
```

Methods	Return
`hs.add(Object o)`	void
`hs.contains(Object o)`	boolean
`hs.remove(Object o)`	boolean
`hs.size()`	int

Set Operations

OPERATIONS WILL MODIFY THE STATE

Let $$ A = {1,2,3} \ B = {3,4,5} $$

```
HashSet<Integer> A = new HashSet<Integer>();
A.add(1);
A.add(2);
A.add(3);

HashSet<Integer> B = new HashSet<Integer>();
```

```
B.add(3);
B.add(4);
B.add(5);
```

Union of A & B $$ A \cup B = { 1,2,3,4,5}$$

```
A.addAll(B) // A new state
```

Intersection of A & B $$ A \cap B = { 3}$$

```
A.retainAll(B) // A new state
```

Difference of A - B (order matters) $$ A - B = {1,2 } $$

```
A.removeAll(B) // A new state
```

$$ B - A = { 4,5 } $$

```
B.removeAll(A) // A new state
```

Symmetric Difference $$A \Delta B = {1,2,3,4} $$

```
HashSet<Type> middle = Intersect(A,B);
HashSet<Type> cloneA =

Symmetric Difference = ()
```

Copy/Clone a set

```
CS210.addAll(cs211); // don't do this. you'll modify the original one


// COPYING
Set<T> temp = new HashSet<>(CS210);
HashSet<String> temp = new HashSet<>(CS210);

HashSet<String> temp = new HashSet<>();
temp.addall(CS211);
```

```
// CLONING

Set<T> temp = (Set<T>CS210.clone());
temp = (HashSet) CS210.clone();
// downcast from Object to Hashset (advanced) leads to unchecked warning
// Hashset.clone() creates a SHALLOW COPY. REFERENCE
```

Hashmaps

```
import java.util.HashMap;

HashMap<keyType, valueType> hs = new HashMap<>();
hs.put(1, "my balls hurt");

System.out.print(hs);
/*
* {1=my balls hurt} // hashmap
* [my balls hurt] // arraylist
* [my balls hurt] // hashset
*
*/

System.out.print(hm.size()); // 1
System.out.print(hm.get(1)); // my balls hurt
```

Methods	**Return**
`hm.put(KeyType a, ValueType b)`	ValueType
`hm.get(Object key)`	ValueType
`hm.size()`	int
`hm.containsKey(Object o)`	void
`hm.containsValue(Object o)`	void
`hm.clear()`	void
`hm.remove(Object o)`	ValueType
`hm.remove(Object key, Object value)`	boolean
`hm.keySet()`	Set
`hm.values()`	Set