ArrayLists

↳ import java.util.*;

```
ArrayList<Type> name = new ArrayList<Type>();
                       ← .ul();
```

ArrayList Methods

- add(value) - appends to end
- add(index, value) - inserts given value JUST BEFORE given index, shifting everything right
- clear() - removes all elements of the list
- indexOf(value) - returns first index where given value is found (-1 if not found)
- get(index) - returns value at given index
- remove(index) - removes value at given index, shifting subsequent values left
- set(index, value) - replaces value at given index with given value
- size() - returns number of elements
- toString() - returns a string representation of the list such as "[3, 42, -7, 15]"
- addAll(list) - adds all elements from the given list to this list (at the end)
- addAll(index, list) - like above, but inserts them at given index
- contains(value) - returns true if given value is found somewhere in the list
- containsAll(list) - returns true if this list contains every element from given list
- equals(list) - returns true if given other list contains same elements
- lastIndexOf(value) - returns last index of value if found in list (-1 if not found)
- remove(value) - finds and removes given value from list
- removeAll(list) - removes any elements found in given list from this list
- retainAll(list) - removes any elements not found in given list from this list
- subList(from, to) - returns the sub-portion of the list between [from and to)
- toArray() - returns elements in this list as an array

ArrayList as parameter: public static void name(ArrayList<Type> name){...}
ArrayList as return: public static ArrayList<Type> name(params){...}

IMPORTANT: The Type you specify must be an object type; it cannot be primitive.
↳ We can still use ArrayLists with primitive types by using wrapper classes

Primitive	Wrapper
int	Integer
double	Double
char	Character
boolean	Boolean

↳ compareTo method
For Strings, A.compareTo(B);
- <0 - A comes "before" B
- >0 - A comes "after" B
- 0 - A "equals" B

eg.
```
String a = "Alice";
String b = "Bob";
if(a.compareTo(b) < 0){ //True
}
```

PRIMITIVES TO OBJECTS

a<b	a.compareTo(b)<0	a!=b	a.compareTo(b)!=0
a<=b	<=0	a>=b	>=0
a==b	==0	a>b	>0

Legal Indexes
↳ 0, list.size()-1
→ Otherwise, IndexOutOfBoundsException

Arrays

`type[] name = new type[length];` ← Each element initially gets a "zero-equivalent" value. int-0, double-0.0, boolean-false, object-null

Legal indexes: 0 - length-1, otherwise - ArrayIndexOutOfBoundsException

An array's length FIELD stores its number of elements : `name.length`

Quick initialization: `type[] name = { value, value, ... value};`
↳ The compiler figures out size

Array Limitations - cannot resize, cannot compare with == or .equals(), cannot print itself

The Arrays class (java.util) has useful static methods for manipulating arrays:
`Arrays.methodName(parameters)`

- `binarySearch(array, value)` - returns index of the given value in sorted array (<0 if not found)
- `copyOf(array, length)` - returns a new copy of array
- `equals(array1, array2)` - true if two arrays contain same elements in same order
- `fill(array, value)` - sets every element to given value
- `sort(array)` - arranges items in a sorted order
- `toString(array)` - returns string representing the array like "[1, 2, 10]"

Array as parameter: `public static void name(type[] n){...}`
Array as return : `public static type[] name(parameters){...}`, `type[] hi = name(params);`

Value semantics - values are copied when assigned, passed as parameters, or returned.
- applies to primitive types

Reference semantics - variables store address of an object in memory (for efficiency)
- when one variable is assigned to the other, the object is NOT copied, both variables refer to the SAME object.
- Modifying the value of one variable will affect others.

Arrays are passed in by reference to fns. Changes made in method are seen everywhere.

```
→ int[] a1 = {4, 15, 8};
  int[] a2 = a1;
  a2[0] = 7;
  System.out.println(Arrays.toString(a1)); //[7, 15, 8]
```

Array

```
String[] names = new String[5];
names[0] = "Bob";
String s = names[0];
```

ArrayList

```
ArrayList<String> names = new ArrayList<String>();
names.add("Bob");
String s = names.get(0);
```

```
ArrayList<int> hi = new ArrayList<int>();
```

* indexOf(c) (c, int i)

Arrays(arr, other params)

ww(index, other) ①

	String	Array	ArrayList
int	.length()	.length	.size()
String		Arrays.toString(array)	.toString()
type	.charAt(int i)	[]	.get(int i), .set(index, value)
boolean	.equals(s), .equalsIgnoreCase(s)	Arrays.equals(arr1, arr2)	.equals(list)
int	.indexOf(c)		.indexOf(c)
String	.trim()		
String	.valueOf()		
String, ArrayList	.substring(int s, int e)		.subList(int s, int e) [s,t)
String, ArrayList	.toUpperCase()		.toArray()
String	.toLowerCase()		
		Arrays.binarySearch(arr, value)	
		Arrays.copyOf(array, length)	
		Arrays.fill(array, value)	
		Arrays.sort(array)	
			.add(val), .add(index, val) ← shift *
			.clear() All *
			.remove(i)
			.set(index, value) *
			.addAll(list), .addAll(index, list)
boolean			.contains(value), .containsAll(list)
			.remove(value), .removeAll(list)
<0 - "str "b4" "s"			.retainAll(list)
>0 - "str" after "s"			
0 - equal	.compareTo(s)		

String hi = "hi";

char[] hi = new char[3]; *
hi[0] = 'h';
or char[] hi = "hi"; ?
or char[] hi = {'h', 'i'} *

ArrayList<char> k = new ArrayList<char>