We provide a quick reference sheet here for *ArrayList* methods you are likely to encounter in CSC 130. To see all *ArrayList* methods, view the *ArrayList* API.

Note: An `ArrayList` is a *parameterized type*, meaning that the behavior of its methods is dependent on the specific type of `ArrayList` that has been created. For this reason, this quick reference sheet provides two sets of descriptions. One set of descriptions is specific to `ArrayList<String>`, the most common `ArrayList` that we work with in our class. The other set of descriptions is *generic*, meant to describe any `ArrayList` (including `ArrayList<String>`).

`ArrayList<String>` Quick Reference

Return Type	Method Definition	Method Description
`void`	`add(String s)`	Adds `s` to the end of the list.
`void`	`add(int i, String s)`	Adds `s` to the list at index `i`. For each item in the list that was at an index greater than or equal to `i` before adding `s`, the item is now at an index that is one more than its previous value.
`String`	`get(int i)`	Returns the `String` at index `i` in the list. This method **does not** remove the item from the list.
`String`	`remove(int i)`	Removes the `String` at index `i` in the list and also returns it. For each item in the list that was at an index greater than or equal to `i` before the removal, the item is now at an index that is one less than its previous value.
`int`	`size()`	Returns the number of items in the list.
`void`	`sort(null)`	When the list contains strings all in the same letter case, sorts the list in alphabetical order. In other cases, sorts in lexicographical order.

`ArrayList<Object>` Quick Reference

Return Type	Method Definition	Method Description
`void`	`add(Object x)`	Adds `x` to the end of the list.
`void`	`add(int i, Object x)`	Adds `x` to the list at index `i`. For each item in the list that was at an index greater than or equal to `i` before adding `x`, the item is now at an index that is one more than its previous value.
`Object`	`get(int i)`	Returns the *`Object`* at index `i` in the list. This method **does not** remove the item from the list.
`Object`	`remove(int i)`	Removes the *`Object`* at index `i` in the list and also returns it. For each item in the list that was at an index greater than or equal to `i` before the removal, the item is now at an index that is one less than its previous value.
`int`	`size()`	Returns the number of items in the list.
`void`	`sort(null)`	Sorts the list based on the way its *`Object`* variables are naturally compared.

`ArrayList<String>` Examples

```
// CONSTRUCTOR
ArrayList<String> list = new ArrayList<String>(); // creates a new ArrayList of String objects


// ADD (to end)
list.add("mon"); // list is now [mon]
list.add("wed"); // list is now [mon, wed]
list.add("fri"); // list is now [mon, wed, fri]


// ADD (at index)
list.add(1, "tue"); // list is now [mon, tue, wed, fri]
list.add(3, "thu"); // list is now [mon, tue, wed, thu, fri]
list.add(0, "zip"); // list is now [zip, mon, tue, wed, thu, fri]
list.add(6, "zap"); // list is now [zip, mon, tue, wed, thu, fri, zip]
list.add(3, "zop"); // list is now [zip, mon, tue, zop, wed, thu, fri, zip]


// REMOVE
list.remove(0); // list is now [mon, tue, zop, wed, thu, fri, zip]
list.remove(2); // list is now [mon, tue, wed, thu, fri, zip]
list.remove(5); // list is now [mon, tue, wed, thu, fri]


// GET
String d = list.get(0); // d is now "mon"; list is still [mon, tue, wed, thu, fri]
d = list.get(4);        // d is now "fri"; list is still [mon, tue, wed, thu, fri]
String m = list.get(2); // m is now "wed"; list is still [mon, tue, wed, thu, fri]


// SIZE
int s = list.size(); // s is 5


// SORT
list.sort(null); // list is now [fri, mon, thu, tue, wed]
```