CHEAT SHEET

Constructing Various Collections

```
List<Integer> list = new ArrayList<Integer>();
Queue<Double> queue = new LinkedList<Double>();
Stack<String> stack = new Stack<String>();
Set<String> words = new HashSet<String>();
Map<String, Integer> counts = new TreeMap<String, Integer>();
```

Methods Found in ALL collections (Lists, Stacks, Queues, Sets, Maps)

`clear()`	removes all elements of the collection
`equals(`**collection**`)`	returns `true` if the given other collection contains the same elements
`isEmpty()`	returns `true` if the collection has no elements
`size()`	returns the number of elements in the collection
`toString()`	returns a string representation such as `"[10, -2, 43]"`

Methods Found in both Lists and Sets (ArrayList, LinkedList, HashSet, TreeSet)

`add(`**value**`)`	adds value to collection (appends at end of list)
`contains(`**value**`)`	returns `true` if the given value is found somewhere in this collection
`remove(`**value**`)`	finds and removes the given value from this collection
`removeAll(`**collection**`)`	removes any elements found in the given collection from this one
`retainAll(`**collection**`)`	removes any elements *not* found in the given collection from this one

`List<E>` Methods (10.1)

`add(`**index**`, `**value**`)`	inserts given value at given index, shifting subsequent values right
`indexOf(`**value**`)`	returns first index where given value is found in list (-1 if not found)
`get(`**index**`)`	returns the value at given index
`lastIndexOf(`**value**`)`	returns last index where given value is found in list (-1 if not found)
`remove(`**index**`)`	removes/returns value at given index, shifting subsequent values left
`set(`**index**`, `**value**`)`	replaces value at given index with given value
`subList(`**from**`, `**to**`)`	returns sub-portion at indexes **from** (inclusive) and **to** (exclusive)

`Stack<E>` Methods

`peek()`	returns the top value from the stack without removing it
`pop()`	removes the top value from the stack and returns it; `peek`/`pop` throw an `EmptyStackException` if the stack is empty
`push(`**value**`)`	places the given value on top of the stack

`Queue<E>` Methods

`add(`**value**`)`	places the given value at the back of the queue
`peek()`	returns the front value from the queue without removing it; returns `null` if the queue is empty
`remove()`	removes the value from the front of the queue and returns it; throws a `NoSuchElementException` if the queue is empty

CHEAT SHEET

`Map<K, V>` Methods (11.3)

`containsKey(`**key**`)`	`true` if the map contains a mapping for the given key
`get(`**key**`)`	the value mapped to the given key (`null` if none)
`keySet()`	returns a `Set` of all keys in the map
`put(`**key**`, `**value**`)`	adds a mapping from the given key to the given value
`putAll(`**map**`)`	adds all key/value pairs from the given map to this map
`remove(`**key**`)`	removes any existing mapping for the given key
`toString()`	returns a string such as `"{a=90, d=60, c=70}"`
`values()`	returns a `Collection` of all values in the map

`String` Methods (3.3, 4.4)

`charAt(`**i**`)`	the character in this String at a given index
`contains(`**str**`)`	`true` if this String contains the other's characters inside it
`endsWith(`**str**`)`	`true` if this String ends with the other's characters
`equals(`**str**`)`	`true` if this String is the same as *str*
`equalsIgnoreCase(`**str**`)`	`true` if this String is the same as *str*, ignoring capitalization
`indexOf(`**str**`)`	first index in this String where given String begins (-1 if not found)
`lastIndexOf(`**str**`)`	last index in this String where given String begins (-1 if not found)
`length()`	number of characters in this String
`startsWith(`**str**`)`	`true` if this String begins with the other's characters
`substring(`**i**`, `**j**`)`	characters in this String from index *i* (inclusive) to *j* (exclusive)
`toLowerCase()`, `toUpperCase()`	a new String with all lowercase or uppercase letters

`Random` Methods (5.1)

`nextBoolean()`	random `true`/`false` result
`nextDouble()`	random real number between 0.0 and 1.0
`nextInt()`	random integer
`nextInt(`**max**`)`	random integer between 0 and *max*

```
public class IntTreeNode {
    public int data;          // data stored in this node
    public IntTreeNode left;  // reference to left subtree
    public IntTreeNode right; // reference to right subtree
    public IntTreeNode(int data) { ... }
    public IntTreeNode(int data, IntTreeNode left, IntTreeNode right) {...}
}
public class IntTree {
    private IntTreeNode overallRoot;
    methods
}
public class ListNode {
    public int data;
    public ListNode Next;
}
public class LinkedIntList {
    private ListNode front;
    methods
}
```