

CSE143 Cheat Sheet

Math Methods (3.2) *mathematical operations*

<code>Math.abs(value)</code>	absolute value
<code>Math.min(v1, v2)</code>	smaller of two values
<code>Math.max(v1, v2)</code>	larger of two values
<code>Math.round(value)</code>	nearest whole number
<code>Math.pow(b, e)</code>	b to the e power

Stacks and Queues (14.2)

(LIFO and FIFO structures)

Queues should be constructed using the `Queue<E>` interface and the `LinkedList<E>` implementation (you may not pass any arguments to the constructor). For example, to construct a queue of String values, you would say:

```
Queue<String> q = new LinkedList<String>();
```

Stacks should be constructed using the `Stack<E>` class (there is no interface):

```
Stack<Integer> s = new Stack<Integer>();
```

For `Stack<E>`, you are limited to the following operations (no iterator or foreach loop):

<code>push(value)</code>	pushes the given value onto the top of the stack
<code>pop()</code>	removes and returns the top of the stack
<code>isEmpty()</code>	returns <code>true</code> if this stack is empty
<code>size()</code>	returns the number of elements in the stack

For `Queue<E>`, you are limited to the following operations (no iterator or foreach loop):

<code>add(value)</code>	adds the given value at the end of the queue
<code>remove()</code>	removes and returns the front of the queue
<code>isEmpty()</code>	returns <code>true</code> if this queue is empty
<code>size()</code>	returns the number of elements in the queue

Iterator<E> Methods (11.1)

(An object that lets you examine the contents of any collection)

<code>hasNext()</code>	returns <code>true</code> if there are more elements to be read from collection
<code>next()</code>	reads and returns the next element from the collection
<code>remove()</code>	removes the last element returned by <code>next</code> from the collection

List<E> Methods (10.1)

(An ordered sequence of values)

<code>add(value)</code>	appends value at end of list
<code>add(index, value)</code>	inserts given value at given index, shifting subsequent values right
<code>clear()</code>	removes all elements of the list
<code>indexOf(value)</code>	returns first index where given value is found in list (-1 if not found)
<code>get(index)</code>	returns the value at given index
<code>remove(index)</code>	removes/returns value at given index, shifting subsequent values left
<code>set(index, value)</code>	replaces value at given index with given value
<code>size()</code>	returns the number of elements in list
<code>isEmpty()</code>	returns <code>true</code> if the list's size is 0
<code>addAll(collection)</code>	adds all elements from the given collection to the end of the list
<code>contains(value)</code>	returns <code>true</code> if the given value is found somewhere in this list
<code>remove(value)</code>	finds and removes the given value from this list if it is present
<code>removeAll(list)</code>	removes any elements found in the given collection from this list
<code>iterator()</code>	returns an object used to examine the contents of the list

Set<E> Methods (11.2)*(A fast-searchable set of unique values)*

add(value)	adds the given value to the set
contains(value)	returns true if the given value is found in the set
remove(value)	removes the given value from the set if it is present
clear()	removes all elements of the set
size()	returns the number of elements in the set
isEmpty()	returns true if the set's size is 0
addAll(collection)	adds all elements from the given collection to the set
containsAll(collection)	returns true if set contains every element from given collection
removeAll(collection)	removes any elements found in the given collection from this set
retainAll(collection)	removes any elements <i>not</i> found in the given collection from this set
iterator()	returns an object used to examine the contents of the set

Map<K, V> Methods (11.3)*(A fast mapping between a set of keys and a set of values)*

put(key, value)	adds a mapping from the given key to the given value
get(key)	returns the value mapped to the given key (null if none)
containsKey(key)	returns true if the map contains a mapping for the given key
remove(key)	removes any existing mapping for the given key
clear()	removes all key/value pairs from the map
size()	returns the number of key/value pairs in the map
isEmpty()	returns true if the map's size is 0
keySet()	returns a Set of all keys in the map
values()	returns a Collection of all values in the map
putAll(map)	adds all key/value pairs from the given map to this map

Point Methods (8.1)*(an object for storing integer x/y coordinates)*

Point(x, y)	constructs a new point with given x/y coordinates
Point()	constructs a new point with coordinates (0, 0)
getX()	returns the x-coordinate of this point
getY()	returns the y-coordinate of this point
equals(other)	returns true if this Point stores the same x/y values as the other
translate(dx, dy)	translates the coordinates by the given amount

String Methods (3.3)*(An object for storing a sequence of characters)*

length()	returns the number of characters in the string
charAt(index)	returns the character at a specific index
equals(other)	returns true if this string equals the other
toUpperCase()	returns a new string with all uppercase letters
toLowerCase()	returns a new string with all lowercase letters
startsWith(other)	returns true if this string starts with the given text
substring(start, stop)	returns a new string composed of character from start index (inclusive) to stop index (exclusive)
substring(start)	returns a new string composed of characters from start index (inclusive) to the end of the string

Collections Implementations

List<E>	ArrayList<E> and LinkedList<E>
Set<E>	HashSet<E> and TreeSet<E> (values ordered)
Map<K, V>	HashMap<K, V> and TreeMap<K, V> (keys ordered)