CSE 143: Computer Programming II — Spring 2015

Midterm Exam Cheat Sheet

Constructing Collections

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

List<E> Methods

`add(`**value**`)`	appends **value** at end of list
`add(`**index**`, `**value**`)`	inserts given **value** at given **index**, shifting subsequent values right
`contains(`**value**`)`	returns `true` if the given **value** is found in the collection
`remove(`**index**`)`	removes/returns value at given **index**, shifting subsequent values left
`get(`**index**`)`	returns the value at given **index**
`set(`**index**`, `**value**`)`	replaces data at given **index** with given **value**
`indexOf(`**value**`)`	returns first index where given **value** is found in list (-1 if not found)
`clear()`	removes all elements of the list
`size()`	returns the number of elements in list
`isEmpty()`	returns true if the list has no elements
`toString()`	returns a string representation of the list such as "`[10, -2, 43]`"
`equals(`**list**`)`	returns true if given **list** contains the same elements

Stack<E> Methods

`push(`**value**`)`	places the given **value** on top of the stack
`pop()`	removes the top value from the stack and returns it; throws a `NoSuchElementException` if the stack is empty
`peek()`	returns the top value from the stack without removing it; throws a `NoSuchElementException` if the stack is empty
`size()`	returns the number of elements in the stack
`isEmpty()`	returns true if the stack has no elements

Queue<E> Methods

`enqueue(`**value**`)`	places the given **value** at the back of the queue
`dequeue()`	removes the value from the front of the queue and returns it; throws a `NoSuchElementException` if the queue is empty
`peek()`	returns the front value from the queue without removing it; throws a NoSuchElementException if the queue is empty
`size()`	returns the number of elements in the queue
`isEmpty()`	returns true if the queue has no elements

`Set<E>` Methods

`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
`clear()`	removes all elements of the set
`size()`	returns the number of elements in the set
`isEmpty()`	returns `true` if there are no elements in the set
`toString()`	returns a `String` representation of the set's elements such as "`[1, 2, 3]`"
`equals(`**set**`)`	returns true if given **set** contains the same elements

`Map<K, V>` Methods

`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 from the given **key**
`remove(`**key**`)`	removes any existing mapping for the given **key**
`keySet()`	returns a `Set` of all keys in the map
`values()`	returns a `Collection` of all values in the map
`clear()`	removes all key/value pairs from the map
`size()`	returns the number of key/value paris in the map
`isEmpty()`	returns `true` if there are no key/value pairs
`toString()`	returns a `String` representation of the map such as "`{a=90, d=60, c=70}`"
`equals(`**map**`)`	returns true if given **map** contains the same elements

`String` Methods

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