^_^ CSE 122 Final Exam Reference Sheet ^_^

(DO NOT WRITE ANY WORK YOU WANTED GRADED ON THIS REFERENCE SHEET. IT WILL NOT BE GRADED)

Examples of Constructing Various Collections

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

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

`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 a 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)
`addAll(`**collection**`)`	Adds all the values in the given collection to this one
`contains(`**value**`)`	Returns `true` if the given value is found somewhere in this collection
`iterator()`	Returns an Iterator object to traverse the collection's elements
`clear()`	Removes all elements of the 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<Type>` Methods

`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

`Stack<Type>` Methods (only allowed methods plus `size` and `isEmpty`)

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

`Queue<Type>` Methods (only allowed methods plus `size` and `isEmpty`)

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

`Map<KeyType, ValueType>` Methods

`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

`Iterator<Type>` Methods

`hasNext()`	Returns `true` if there is another element in the iterator
`next()`	Returns the next value in the iterator and progresses the iterator forward one element
`remove()`	Removes the previous value returned by the next. Can only call once after each call to `next()`

`String` Methods

`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
`isEmpty()`	`true` if this String is the empty 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)
`substring(`**i**`)`	Characters in this String from index *i* (inclusive) to the end
`toLowerCase()`, `toUpperCase()`	A new String with all lowercase or uppercase letters

`Math` Methods

`abs(`**x**`)`	Returns the absolute value of `x`
`max(`**x, y**`)`	Returns the larger of `x` and `y`
`min(`**x, y**`)`	Returns the smaller of `x` and `y`
`pow(`**x, y**`)`	Returns the value of `x` to the `y` power
`random()`	Returns a random number between `0.0` and `1.0`
`round(`**x**`)`	Returns `x` rounded to the nearest integer

Object/Interface Syntax

```
public class Example implements InterfaceExample {
    private type field;
    public Example() {
       field = something;
    }
    public void method() {
       // do something
    }
}
```

```
public interface InterfaceExample {
    public void method();
}
```