CSE 123 Final Exam Reference Sheet

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

Methods Found in ALL collections (`List`, `Set`, `Map`)

`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 a collection
`toString()`	Returns a string representation such as `"[10, -2, 43]"`

Methods Found in both `List` and `Set` (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
`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

`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

`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

`String` Methods

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

Inheritance Syntax

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

public abstract class AbstractExample {
    private type field;

    public void method() {
       // do something
    }

    public abstract void abstractMethod();
}

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

`LinkedIntList` **Class**

```
public class LinkedIntList {
    private ListNode front;
    private int size;

    public LinkedIntList() {
        ...
    }

    public LinkedIntList(int[] nums) {
        ...
    }

    public void add(int index, int value) {
        ...
    }

    public void add(int value) {
        ...
    }

    public int size() {
        ...
    }

    public int get(int index) {
        ...
    }

    public static class ListNode {
        public final int data;
        public ListNode next;

        public ListNode(int data) {
            this(data, null);
        }

        public ListNode(int data, ListNode next) {
            this.data = data;
            this.next = next;
        }
    }
}
```

`IntTree` Class

```
public class IntTree {
    private IntTreeNode overallRoot;

    public IntTree() {
        ...
    }

    public boolean contains(int value) {
        ...
    }

    public void add(int value) {
        ...
    }


    private static class IntTreeNode {
        public final int data;
        public IntTreeNode left;
        public IntTreeNode right;

        public IntTreeNode(int data) {
            this(data, null, null);
        }

        public IntTreeNode(int data, IntTreeNode left, IntTreeNode right) {
            this.data = data;
            this.left = left;
            this.right = right;
        }
    }
}
```