

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)

<code>clear()</code>	Removes all elements of the collection
<code>equals(collection)</code>	Returns <code>true</code> if the given other collection contains the same elements
<code>isEmpty()</code>	Returns <code>true</code> if the collection has no elements
<code>size()</code>	Returns the number of elements in a collection
<code>toString()</code>	Returns a string representation such as "[10, -2, 43]"

Methods Found in both List and Set (ArrayList, LinkedList, HashSet, TreeSet)

<code>add(value)</code>	Adds value to collection (appends at end of list)
<code>addAll(collection)</code>	Adds all the values in the given collection to this one
<code>contains(value)</code>	Returns <code>true</code> if the given value is found somewhere in this collection
<code>iterator()</code>	Returns an Iterator object to traverse the collection's elements
<code>remove(value)</code>	Finds and removes the given value from this collection
<code>removeAll(collection)</code>	Removes any elements found in the given collection from this one
<code>retainAll(collection)</code>	Removes any elements <i>not</i> found in the given collection from this one

List<Type> Methods

<code>add(index, value)</code>	Inserts given value at given index, shifting subsequent values right
<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>lastIndexOf(value)</code>	Returns last index where given value is found in list (-1 if not found)
<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

Map<KeyType, ValueType> Methods

<code>containsKey(key)</code>	<code>true</code> if the map contains a mapping for the given key
<code>get(key)</code>	The value mapped to the given key (<code>null</code> if none)
<code>keySet()</code>	Returns a <code>Set</code> of all keys in the map
<code>put(key, value)</code>	Adds a mapping from the given key to the given value
<code>putAll(map)</code>	Adds all key/value pairs from the given map to this map
<code>remove(key)</code>	Removes any existing mapping for the given key
<code>toString()</code>	Returns a string such as "{l=90, d=60, c=70}"
<code>values()</code>	Returns a <code>Collection</code> of all values in the map

Math Methods

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

String Methods

<code>charAt(i)</code>	Returns the character in this String at a given index
<code>contains(str)</code>	Returns <code>true</code> if this String contains the other's characters inside it
<code>endsWith(str)</code>	Returns <code>true</code> if this String ends with the other's characters
<code>equals(str)</code>	Returns <code>true</code> if this String is the same as <i>str</i>
<code>equalsIgnoreCase(str)</code>	Returns <code>true</code> if this String is the same as <i>str</i> , ignoring capitalization
<code>indexOf(str)</code>	Returns the first index in this String where <i>str</i> begins (-1 if not found)
<code>lastIndexOf(str)</code>	Returns the last index in this String where <i>str</i> begins (-1 if not found)
<code>length()</code>	Returns the number of characters in this String
<code>isEmpty()</code>	Returns <code>true</code> if this String is the empty string
<code>startsWith(str)</code>	Returns <code>true</code> if this String begins with the other's characters
<code>substring(i, j)</code>	Returns the characters in this String from index <i>i</i> (inclusive) to <i>j</i> (exclusive)
<code>substring(i)</code>	Returns the characters in this String from index <i>i</i> (inclusive) to the end
<code>toLowerCase()</code>	Returns a new String with all this String's letters changed to lowercase
<code>toUpperCase()</code>	Returns a new String with all this String's letters changed to uppercase
<code>compareTo(str)</code>	Returns a negative number if this comes lexicographically (alphabetically) before other, 0 if they're the same, positive if this comes lexicographically after other.

JUnit Methods

<code>assertEquals(expected, actual)</code>	Tests that expected equals actual (using <code>.equals</code>)
<code>assertNotEquals(expected, actual)</code>	Tests that expected doesn't equal actual (using <code>.equals</code>)
<code>assertSame(expected, actual)</code>	Tests that expected equals actual (using <code>==</code>)
<code>assertNotSame(expected, actual)</code>	Tests that expected doesn't equal actual (using <code>==</code>)
<code>assertTrue(actual)</code>	Tests that actual is true
<code>assertFalse(actual)</code>	Tests that actual is false

Abstract Class Syntax

```
public abstract class AbstractClass{
    // an abstract class can contain fields
    private type name;

    // an abstract class can contain constructors
    public AbstractClass(...) {
        // initialize the object
    }

    public abstract returnType abstractMethod(...);

    public returnType implementedMethod(...) {
        ...
    }
}
```

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();
}
```

IntList Interface

```
public interface IntList {
    public void add(int value);
    public void add(int index, int value);
    public void remove(int index);
    public void set(int index, int value);
    public int get(int index);
    public int size();
    public int indexOf(int value);
    public boolean contains(int value);
    public String toString();
}
```

ArrayIntList Class

```
public class ArrayIntList implements IntList {
    private int[] elementData;
    private int size;

    public static final int DEFAULT_CAPACITY = 10;

    public ArrayIntList() {...}

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

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

    public void remove(int index) {...}

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

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

    public int size() {...}

    public int indexOf(int value) {...}

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

    public String toString() {...}
}
```

LinkedList Class

```
public class LinkedList implements IntList {
    private ListNode front;

    public LinkedList() {...}

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

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

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

    public void remove(int index) {...}

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

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

    public int size() {...}

    public int indexOf(int value) {...}

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

    public String toString() {...}

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

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

IntTree Class

```
public class IntTree {
    private IntTreeNode overallRoot;

    public IntTree() {...}
    public IntTree(int[] arr) {...}
    public boolean contains(int value) {...}
    public String toString() {...}

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

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

This page intentionally left blank