Linked Lists (16.2)
Below is an example of a method that could be added to the LinkedList class to compute the sum of the list:

```java
class LinkedList {
    public int sum() {
        int sum = 0;
        ListNode current = front;
        while (current != null) {
            sum += current.data;
            current = current.next;
        }
        return sum;
    }
}
```

Math Methods (3.2)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.abs(value)</td>
<td>absolute value</td>
</tr>
<tr>
<td>Math.min(v1, v2)</td>
<td>smaller of two values</td>
</tr>
<tr>
<td>Math.max(v1, v2)</td>
<td>larger of two values</td>
</tr>
<tr>
<td>Math.round(value)</td>
<td>nearest whole number</td>
</tr>
<tr>
<td>Math.pow(b, e)</td>
<td>b to the e power</td>
</tr>
</tbody>
</table>

Stacks and Queues (14.2)

Queues should be constructed using the Queue<E> interface and the LinkedList<E> implementation. For example, to construct a queue of String values, you would say:

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

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

```java
Stack<String> s = new Stack<String>();
```

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>push(value)</td>
<td>pushes the given value onto the top of the stack</td>
</tr>
<tr>
<td>pop()</td>
<td>removes and returns the top of the stack</td>
</tr>
<tr>
<td>isEmpty()</td>
<td>returns true if this stack is empty</td>
</tr>
<tr>
<td>size()</td>
<td>returns the number of elements in the stack</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(value)</td>
<td>adds the given value at the end of the queue</td>
</tr>
<tr>
<td>remove()</td>
<td>removes and returns the front of the queue</td>
</tr>
<tr>
<td>isEmpty()</td>
<td>returns true if this queue is empty</td>
</tr>
<tr>
<td>size()</td>
<td>returns the number of elements in the queue</td>
</tr>
</tbody>
</table>

Iterator<E> Methods (11.1)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasNext()</td>
<td>returns true if there are more elements to be read from collection</td>
</tr>
<tr>
<td>next()</td>
<td>reads and returns the next element from the collection</td>
</tr>
<tr>
<td>remove()</td>
<td>removes the last element returned by next from the collection</td>
</tr>
</tbody>
</table>

List<E> Methods (10.1)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(value)</td>
<td>appends value at end of list</td>
</tr>
<tr>
<td>add(index, value)</td>
<td>inserts given value at given index, shifting subsequent values right</td>
</tr>
<tr>
<td>clear()</td>
<td>removes all elements of the list</td>
</tr>
<tr>
<td>indexOf(value)</td>
<td>returns first index where given value is found in list (-1 if not found)</td>
</tr>
<tr>
<td>get(index)</td>
<td>returns the value at given index</td>
</tr>
<tr>
<td>remove(index)</td>
<td>removes/returns value at given index, shifting subsequent values left</td>
</tr>
<tr>
<td>set(index, value)</td>
<td>replaces value at given index with given value</td>
</tr>
<tr>
<td>size()</td>
<td>returns the number of elements in list</td>
</tr>
<tr>
<td>isEmpty()</td>
<td>returns true if the list’s size is 0</td>
</tr>
<tr>
<td>addAll(collection)</td>
<td>adds all elements from the given collection to the end of the list</td>
</tr>
<tr>
<td>contains(value)</td>
<td>returns true if the given value is found somewhere in this list</td>
</tr>
<tr>
<td>remove(value)</td>
<td>finds and removes the given value from this list</td>
</tr>
<tr>
<td>removeAll(list)</td>
<td>removes any elements found in the given collection from this list</td>
</tr>
<tr>
<td>iterator()</td>
<td>returns an object used to examine the contents of the list</td>
</tr>
</tbody>
</table>
### Set<E> Methods (11.2)  
*(A fast-searchable set of unique values)*

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

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

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

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point(x, y)</td>
<td>constructs a new point with given x/y coordinates</td>
</tr>
<tr>
<td>Point()</td>
<td>constructs a new point with coordinates (0, 0)</td>
</tr>
<tr>
<td>getX()</td>
<td>returns the x-coordinate of this point</td>
</tr>
<tr>
<td>getY()</td>
<td>returns the y-coordinate of this point</td>
</tr>
<tr>
<td>equals(other)</td>
<td>returns true if this Point stores the same x/y values as the other</td>
</tr>
<tr>
<td>translate(dx, dy)</td>
<td>translates the coordinates by the given amount</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length()</td>
<td>returns the number of characters in the string</td>
</tr>
<tr>
<td>charAt(index)</td>
<td>returns the character at a specific index</td>
</tr>
<tr>
<td>compareTo(other)</td>
<td>returns how this string compares to the other</td>
</tr>
<tr>
<td>equals(other)</td>
<td>returns true if this string equals the other</td>
</tr>
<tr>
<td>toUpperCase()</td>
<td>returns a new string with all uppercase letters</td>
</tr>
<tr>
<td>toLowerCase()</td>
<td>returns a new string with all lowercase letters</td>
</tr>
<tr>
<td>startsWith(other)</td>
<td>returns true if this string starts with the given text</td>
</tr>
<tr>
<td>substring(start, stop)</td>
<td>returns a new string composed of character from start index (inclusive) to stop index (exclusive)</td>
</tr>
</tbody>
</table>

### Collections Implementations

<table>
<thead>
<tr>
<th>Collection</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>List&lt;E&gt;</td>
<td>ArrayList&lt;E&gt; and LinkedList&lt;E&gt;</td>
</tr>
<tr>
<td>Set&lt;E&gt;</td>
<td>HashSet&lt;E&gt; and TreeSet&lt;E&gt; <em>(values ordered)</em></td>
</tr>
<tr>
<td>Map&lt;K, V&gt;</td>
<td>HashMap&lt;K, V&gt; and TreeMap&lt;K, V&gt; <em>(keys ordered)</em></td>
</tr>
</tbody>
</table>