Java Quick Reference

Accessible methods from the Java library that may be included in the test

|  |  |
| --- | --- |
| **Class Constructors and Methods** | **Explanation** |
| **LinkedList<E> Class** | |
| **LinkedList**() | Constructs an empty list. |
| void **addFirst**(E e) | Inserts the specified element at the beginning of this list. |
| void **addLast**(E e) boolean **add**(E e) | Appends the specified element to the end of this list. **add()** always returns true. |
| E **get**(int index) | Returns the element at the specified position in this list. Throws **IndexOutOfBoundsException** if index is out of range (index < 0 || index >= size()) |
| E **getFirst**() | Returns the first element in this list. Throws **NoSuchElementException** if the list is empty. |
| E **getLast**() | Returns the last element in this list. Throws **NoSuchElementException** if the list is empty. |
| E **removeFirst**() E **remove**() | Retrieves and removes the head (first element) of this list. Throws **NoSuchElementException** if the list is empty. |
| E **removeLast**() | Retrieves and removes and returns the last element from this list. Throws **NoSuchElementException** if the list is empty. |
| E **remove**(int index) | Removes the element at the specified position in this list. Shifts any subsequent elements to the left (subtracts one from their indices). Returns the element that was removed from the list. Throws **IndexOutOfBoundsException** if index is out of range (index < 0 || index >= size()) |
| E **peekFirst**() E **peek**() | Retrieves, but does not remove, the first element of this list, or returns null if this list is empty. |
| E **peekLast**() | Retrieves, but does not remove, the last element of this list, or returns null if this list is empty. |
| int **size**() | Returns the number of elements in this list. |
| **Stack<E> Class** | |
| boolean **empty**() | Tests if this stack is empty. |
| E **peek**() | Looks at the object at the top of this stack without removing it from the stack. Throws **EmptyStackException** if the stack is empty. |
| E **pop**() | Removes the object at the top of this stack and returns that object as the value of this function. Throws **EmptyStackException** if the stack is empty. |
| E **push**(E item) | Pushes an item onto the top of this stack. |
| int **size**() | Returns the number of items in this stack. |
| **Queue<E> Interface** | |
| boolean **add**(E e) | Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an **IllegalStateException** if no space is currently available. |
| E **remove**() | Retrieves and removes the head of this queue. Throws **NoSuchElementException** if the queue is empty. |
| E **peek**() | Retrieves, but does not remove, the head of this queue, or returns **null** if this queue is empty. |

|  |  |
| --- | --- |
| **Class Constructors and Methods** | **Explanation** |
| **HashMap<K, V> Class** | |
| **HashMap**() | Constructs an empty HashMap with the default initial capacity (16) and the default load factor (0.75).. |
| **HashMap**(Map<K,V> m) | Constructs a new HashMap with the same mappings as the specified Map. |
| V **put**(K key, V value) | Associates the specified value with the specified key in this map. Returns: the previous value associated with key, or null if there was none. |
| V **get**(Object key) | Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key. |
| boolean **containsKey**(Object key) | Returns true if this map contains a mapping for the specified key. |
| boolean **containsValue**(Object value) | Returns true if this map maps one or more keys to the value. |
| Set<Map.Entry<K, V>> **entrySet**() | Returns a Set view of the mappings contained in this map. |
| Set<K> **keySet**() | Returns a Set view of the keys contained in this map. |
| V **remove** (Object key) | Removes the mapping for the specified key from this map if present. Returns the previous value associated with the key or null if there was none. |
| void **clear**() | Removes all of the mappings from this map. |
| int **size**() | Returns the number of key-value mappings in this map. |
| **Map.Entry<K, V> Interaface** | |
| K **getKey**() | Returns the key corresponding to this entry. |
| V **getValue**() | Returns the value corresponding to this entry. |
| boolean **equals**(Object o) | Returns true if the given object is also a map entry and the two entries represent the same mapping |
| **Set<E> Interface** | |
| boolean **contains**(Object o) | Returns true if this set contains the specified element. |
| boolean **isEmpty**() | Returns true if this set contains no elements. |
| int **size**() | Returns the number of elements in this set (its cardinality). |
| boolean **remove**(Object o) | Removes the specified element from this set if it is present. Returns true if this set contained the specified element. |
| void **clear**() | Removes all of the elements from this set. |
| Object[] **toArray**() | Returns an array containing all of the elements in this set. |

Code sample: Iterating through all objects in a HashMap<K,V> collection

HashMap<K,V> hashMap = new HashMap<K, V>();

K k = /\* some key \*/

V v = /\* some value \*/

hashMap.**put**(k, v);

/\* … \*/

**for**(Map.Entry<K,V> mapEntry : hashMap.**entrySet**()) {

K key = mapEntry.**getKey**();

V value = mapEntry.**getValue**();

/\* … \*/

}