Collections

Methods

Contains

it's `Object`, not the type of the Collection!

```
boolean contains(Object o)
```

removeIf

```
List<String> list = new ArrayList<>(List.of("ab", "bb", "cb"));
boolean status = list.removeIf(s -> s.startsWith("a"));
System.out.println(status); // true
System.out.println(list); //["bb', "cb"]
```

Immutable Collections

```
List<String> list = List.of("a", "b");
Set<String> set = Set.of("a", "b");  //accepts a vararg
List<String> listCopyOf = List.copyOf(list);  //accepts a Collectio
List<String> listCopyOfSet = List.copyOf(set);
Set<String> setCopyOf = Set.copyOf(set);
Set<String> setCopyOfList = Set.copyOf(list);
```

Sorting an immutable collection

```
List<Integer> list = List.of(5, 3, 1);
//I cannot sort an immutable collection!
//Exception in thread "main" java.lang.UnsupportedOperationExceptio
//  at java.base/java.util.ImmutableCollections.uoe(ImmutableCollec
Collections.sort(list);
```

TreeSet

```
Comparator<Integer> comparator = (n1,n2)->n1-n2;
TreeSet<Integer> set1 = new TreeSet<>(comparator);
TreeSet<Integer> set2 = new TreeSet<>(Set.of(1, 2, 3));
```

List

remove - mind the overloading of `remove()`

```
//this remove the element at index 2, because here we call remove(i
list.remove(2); //[5,3,1]
//this removes element 1 as here we call remove(Object obj)
list.remove(Integer.valueOf(1)); //[5, 3]
```

Overloading of remove

Creating a List with Factory

List Factory

Comparator

package: `java.util`

```
int compare(T o1, T o2);
```

reversed

```
Comparator<T> reversed()
```

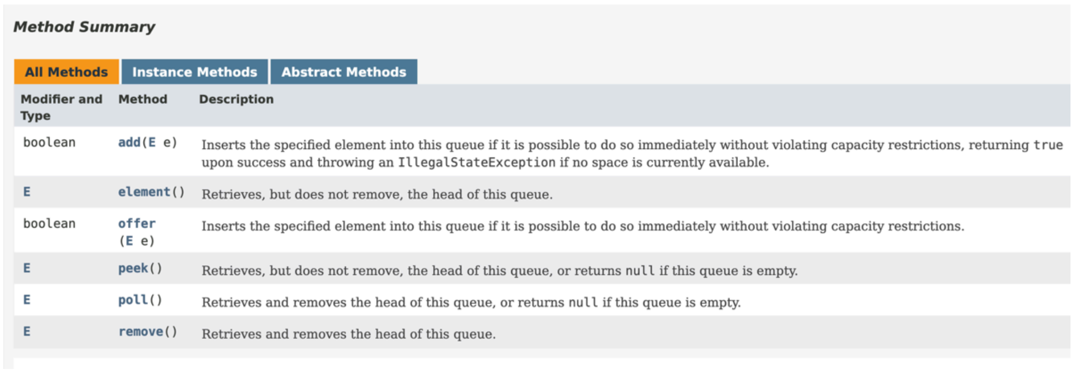
Returns a comparator that imposes the reverse ordering of this comparator.

Comparable

package: `java.lang`

```
int compareTo(T o1);
```

Queue

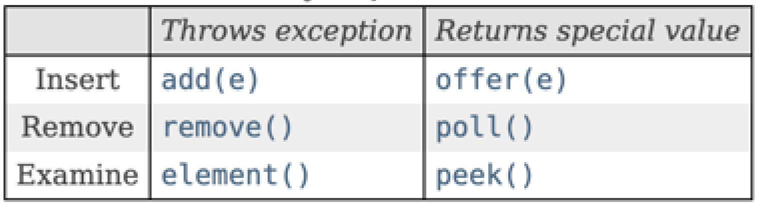


Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method	Description
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	`element()`	Retrieves, but does not remove, the head of this queue.
boolean	`offer(E e)`	Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.
E	`peek()`	Retrieves, but does not remove, the head of this queue, or returns `null` if this queue is empty.
E	`poll()`	Retrieves and removes the head of this queue, or returns `null` if this queue is empty.
E	`remove()`	Retrieves and removes the head of this queue.

Summary of Queue methods



	Throws exception	*Returns special value*
Insert	`add(e)`	`offer(e)`
Remove	`remove()`	`poll()`
Examine	`element()`	`peek()`

Main methods of Queue

The following **throw an exception** if something go wrong:

```
public boolean add(E e);
public E element(); //equivalent to peek()
public E remove();
```

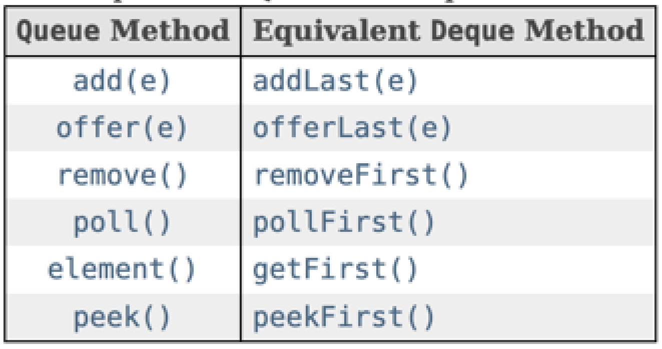
The following **do not throw an exception** if something go wrong:

```
public boolean offer(E e);
public E peek();
public E poll();
```

Deque

- LinkedList implements Deque
- ArrayDeque implements Deque

Comparison of Queue and Deque methods



Queue Method	Equivalent Deque Method
add(e)	addLast(e)
offer(e)	offerLast(e)
remove()	removeFirst()
poll()	pollFirst()
element()	getFirst()
peek()	peekFirst()

Main methods of Deque

The following **throw an exception** if something go wrong:

```
public void addFirst(E e);
public void addLast(E e);
public E getFirst();  //element not removed
public E getLast();  //element not removed
public E removeFirst();
public E removeLast();
```

The following **do not throw an exception** if something go wrong:

```
public boolean offerFirst(E e);
public boolean offerLast(E e);
public E peekFirst();
public E peekLast();
public E pollFirst();
public E pollLast();
```

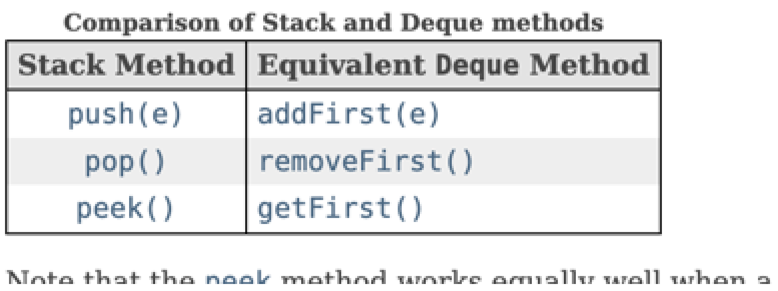
Example Deque

```
//The offer() method inserts an element at the end of the queue
Deque<String> q = new ArrayDeque<>();
q.offer("dog"); // [dog]
q.offer("cat"); // [dog, cat]
q.offer("bunny"); // [dog, cat, bunny]
System.out.print(q.peek() + " " + q.size()); // dog 3
```

```
public interface Deque<E> extends Queue<E> {/**/}
```

DequeAsAStack

Stack



Comparison of Stack and Deque methods

Stack Method	Equivalent Deque Method
push(e)	addFirst(e)
pop()	removeFirst()
peek()	getFirst()

Note that the peek method works equally well when a

pop() vs poll()

When Deque is empty:

- `pop()` throws `java.util.NoSuchElementException`
- `poll()` returns null

```
Deque<String> stack = new LinkedList<>();  //empty deque
String result = stack.poll();  //this returns null
System.out.println(result); //null
//Exception in thread "main" java.util.NoSuchElementException
String pop = stack.pop();
```

Map

foreach

```
    Map<Integer, String> map = buildMap();
    BiConsumer<Integer, String> biConsumer =
        (key, value)-> System.out.println("key: %s - value: %s".forr
    //NOTE! it uses a BIConsumer, not a consumer!
    map.forEach(biConsumer);
```

merge

```
Map<Integer, Integer> map = new HashMap<>();
map.put(1, null);
///If the specified key is not already associated with a value or is
map.merge(1, 4, (v1, v2)->v1+v2);  //[1,4]
```

MapMerge

TreeMap

Keys added to `TreeMap` need to implement `Comparable`, as less as a `Comparator` is provided.

Usage Of TreeMap

```
Map map = new TreeMap<>();
map.put(1, "2");
//java.lang.ClassCastException: class java.lang.Integer cannot be c
map.put("hello", "2");
System.out.println(map);
```

Collections and null values

- `ArrayList`: allows null
- `LinkedList`: allows null
- `HashSet`: allows null
- `TreeSet`: **DOES NOT** allow null

```
Set<Integer> set = new TreeSet<>();
//Exception java.lang.NullPointerException: Cannot invoke "java.lan
set.add(null);
```