✅ Master Table of Methods in `ArrayList`, `LinkedList`, `Vector`, and `Stack`

Method	ArrayList	LinkedList	Vector	Stack
`add(E e)`	✅	✅	✅	✅
`add(int, E)`	✅	✅	✅	✅
`addAll(Collection)`	✅	✅	✅	✅
`addAll(int, Collection)`	✅	✅	✅	✅
`clear()`	✅	✅	✅	✅
`clone()`	✅	✅	✅	✅
`contains(Object)`	✅	✅	✅	✅
`containsAll(Collection)`	✅	✅	✅	✅
`equals(Object)`	✅	✅	✅	✅
`ensureCapacity(int)`	✅	❌	✅	✅
`forEach(Consumer)`	✅	✅	✅	✅
`get(int)`	✅	✅	✅	✅
`indexOf(Object)`	✅	✅	✅	✅
`isEmpty()`	✅	✅	✅	✅
`iterator()`	✅	✅	✅	✅
`lastIndexOf(Object)`	✅	✅	✅	✅
`listIterator()`	✅	✅	✅	✅
`listIterator(int)`	✅	✅	✅	✅
`remove(int)`	✅	✅	✅	✅
`remove(Object)`	✅	✅	✅	✅
`removeAll(Collection)`	✅	✅	✅	✅

`removeIf(Predicate)`	✅	✅	✅	✅
`replaceAll(UnaryOperator)`	✅	✅	✅	✅
`retainAll(Collection)`	✅	✅	✅	✅
`set(int, E)`	✅	✅	✅	✅
`size()`	✅	✅	✅	✅
`sort(Comparator)`	✅	✅	✅	✅
`spliterator()`	✅	✅	✅	✅
`subList(int, int)`	✅	✅	✅	✅
`toArray()`	✅	✅	✅	✅
`toArray(T[])`	✅	✅	✅	✅
`toString()`	✅	✅	✅	✅
`trimToSize()`	✅	❌	✅	✅
`hashCode()`	✅	✅	✅	✅
`addFirst(E)`	❌	✅	❌	❌
`addLast(E)`	❌	✅	❌	❌
`getFirst()`	❌	✅	❌	❌
`getLast()`	❌	✅	❌	❌
`offer(E)`	❌	✅	❌	❌
`offerFirst(E)`	❌	✅	❌	❌
`offerLast(E)`	❌	✅	❌	❌
`peek()`	❌	✅	❌	✅
`peekFirst()`	❌	✅	❌	❌
`peekLast()`	❌	✅	❌	❌
`poll()`	❌	✅	❌	❌

`pollFirst()`	❌	✅	❌	❌
`pollLast()`	❌	✅	❌	❌
`pop()`	❌	✅	❌	✅
`push(E)`	❌	✅	❌	✅
`removeFirst()`	❌	✅	❌	❌
`removeLast()`	❌	✅	❌	❌
`descendingIterator()`	❌	✅	❌	❌
`capacity()`	❌	❌	✅	✅
`copyInto(Object[])`	❌	❌	✅	✅
`elements()`	❌	❌	✅	✅
`firstElement()`	❌	❌	✅	✅
`lastElement()`	❌	❌	✅	✅
`insertElementAt(E, int)`	❌	❌	✅	✅
`removeAllElements()`	❌	❌	✅	✅
`removeElement(Object)`	❌	❌	✅	✅
`removeElementAt(int)`	❌	❌	✅	✅
`setElementAt(E, int)`	❌	❌	✅	✅
`elementAt(int)`	❌	❌	✅	✅
`setSize(int)`	❌	❌	✅	✅
`empty()`	❌	❌	❌	✅
`search(Object)`	❌	❌	❌	✅

✅ **Legend**:

- ✅ = Method exists in that class
- ❌ = Method does not exist

Linked list specific

1. Understanding `addFirst(E)` vs `offerFirst(E)`

Both are methods from the Deque interface implemented by classes like `LinkedList` and `ArrayDeque`.

Method	Description	Throws Exception?	Returns Value
addFirst(E)	Inserts the specified element at the front of the deque.	Throws `IllegalStateException` if no space is available (capacity restricted deque).	Void (no return)
offerFirst(E)	Inserts the specified element at the front of the deque if possible.	Does not throw exception. Returns `false` if no space is available.	`boolean` (true if success, false if failure)

Key difference:

- `addFirst` throws an exception if the operation fails (e.g., capacity restrictions).
- `offerFirst` returns false if the operation fails but does NOT throw an exception.

Code Example to Illustrate:

java
CopyEdit
```
import java.util.ArrayDeque;
import java.util.Deque;

public class AddFirstVsOfferFirst {
    public static void main(String[] args) {
        // Creating a deque with capacity 2 (ArrayDeque has no capacity limit by
default, but for demo)
        Deque<Integer> deque = new ArrayDeque<>(2);

        // Using addFirst()
        deque.addFirst(10);
        deque.addFirst(20);

        // Uncommenting this would throw IllegalStateException if capacity
exceeded:
        // deque.addFirst(30);

        System.out.println("Deque after addFirst: " + deque);

        // Using offerFirst()
        boolean offered1 = deque.offerFirst(30);  // Will return false if capacity
exceeded (not in this case)
        System.out.println("OfferFirst success? " + offered1);
        System.out.println("Deque after offerFirst: " + deque);
    }
}
```

Output:

yaml
CopyEdit
```
Deque after addFirst: [20, 10]
OfferFirst success? true
Deque after offerFirst: [30, 20, 10]
```

---

2. Other related methods and their differences

Method	Description	Throws Exception?	Returns
`addFirst(E)`	Add element at front; throws `IllegalStateException` if fails	Yes	Void
`offerFirst(E)`	Add element at front; returns false if fails	No	boolean
`addLast(E)`	Add element at end; throws exception if fails	Yes	Void
`offerLast(E)`	Add element at end; returns false if fails	No	boolean
`peekFirst()`	Returns first element or null if empty	No	element or null
`peekLast()`	Returns last element or null if empty	No	element or null
`getFirst()`	Returns first element; throws `NoSuchElementException` if empty	Yes	element

`getLast()`	Returns last element; throws `NoSuchElementException` if empty	Yes	element
`pollFirst()`	Retrieves and removes first element or returns null if empty	No	element or null
`pollLast()`	Retrieves and removes last element or returns null if empty	No	element or null
`removeFirst()`	Retrieves and removes first element; throws exception if empty	Yes	element
`removeLast()`	Retrieves and removes last element; throws exception if empty	Yes	element

---

3. Summary of differences (add vs offer, get vs peek, remove vs poll):

Pair	Difference
addX() vs offerX()	`addX()` throws exception if fails; `offerX()` returns false
getX() vs peekX()	`getX()` throws exception if deque empty; `peekX()` returns null
removeX() vs pollX()	`removeX()` throws exception if deque empty; `pollX()` returns null

---

4. Code example covering key methods:

```
import java.util.Deque;
import java.util.LinkedList;
import java.util.NoSuchElementException;

public class DequeMethodsDemo {
    public static void main(String[] args) {
        Deque<String> deque = new LinkedList<>();

        // addFirst vs offerFirst
        deque.addFirst("A"); // throws exception if full (not in LinkedList)
        boolean offerResult = deque.offerFirst("B"); // returns false if can't add

        System.out.println("Deque after addFirst and offerFirst: " + deque); // [B,
A]
```

```
        // peekFirst vs getFirst
        System.out.println("peekFirst: " + deque.peekFirst()); // B (null if empty)
        System.out.println("getFirst: " + deque.getFirst());   // B (throws
exception if empty)

        // pollFirst vs removeFirst
        System.out.println("pollFirst: " + deque.pollFirst());   // removes and
returns B or null
        System.out.println("removeFirst: " + deque.removeFirst()); // removes and
returns A or throws exception

        // peekLast vs getLast
        System.out.println("peekLast: " + deque.peekLast());  // null (deque empty)
        try {
            System.out.println("getLast: " + deque.getLast());  // throws
NoSuchElementException
        } catch (NoSuchElementException e) {
            System.out.println("getLast threw exception: " + e);
        }
    }
}
```

---

5. Explanation of `Stack` vs `Deque` methods

- `Stack` is legacy and extends `Vector`, it does not have `addFirst()` or `offerFirst()`.
- Stack methods are mostly:
  - `push(E)` → push element on top (like `addFirst` on deque)
  - `pop()` → remove and return top element
  - `peek()` → view top element without removing
- Deque is a double-ended queue, more flexible and recommended over `Stack` in modern Java.

Summary Table

Method	Stack	LinkedList (Deque)	Vector	ArrayList
addFirst(E)	❌	✅	❌	❌
offerFirst(E)	❌	✅	❌	❌

getFirst()	❌	✅	❌	❌
peekFirst()	❌	✅	❌	❌
pop()	✅	✅	❌	❌
push(E)	✅	✅	❌	❌
...				

If you want, I can also provide a full table with all method differences explained and code snippets for any specific method. Would you like that?

4.1-mini