Collection Framework Methods:-

Shambhu Kumar

@Kumarsam07

@javac_java

http://linkedin.com/in/kumarsam07

1. Collection Interface Methods:

The Collection interface in Java, part of the java.util package, defines the foundation for all collections in the Java Collections Framework. Below is a list of its methods along with descriptions:

i. **boolean add(E e)**
   Adds the specified element to the collection if it is not already present.

ii. **boolean addAll(Collection<? extends E> c)**
   Adds all the elements from the specified collection to the current collection.

iii. **boolean remove(Object o)**
   Removes a single instance of the specified element from the collection, if it is present.

iv. **boolean removeAll(Collection<?> c)**
   Removes all elements that are also contained in the specified collection.

v. **boolean retainAll(Collection<?> c)**
Retains only the elements in the collection that are contained in the specified collection.

vi. **void clear()**
Removes all elements from the collection.

vii. **boolean contains(Object o)**
Checks if the collection contains the specified element.

viii. **boolean containsAll(Collection<?> c)**
Checks if the collection contains all elements of the specified collection.

ix. **boolean equals(Object o)**
Compares the specified object with the collection for equality.

x. **int hashCode()**
Returns the hash code value for the collection.

xi. **boolean isEmpty()**
Checks if the collection is empty.

xii. **Iterator<E> iterator()**
Returns an iterator over the elements in the collection.

xiii. **int size()**
Returns the number of elements in the collection.

xiv. **Object[] toArray()**
Returns an array containing all the elements in the collection.

xv. **<T> T[] toArray(T[] a)**
Returns an array containing all elements in the collection, using the provided array type.

2. Iterator Interface Methods:

The Iterator interface in Java, part of the java.util package, provides methods for traversing collections in a forward-only direction. It is one of the most fundamental components of the Java Collections Framework. Below are the methods of the Iterator interface with detailed descriptions:

1. **boolean hasNext()**
   - Description: Checks if the iteration has more elements.
   - Returns: true if there are more elements to iterate over, otherwise false.
   - Usage: Typically used as a condition in a while loop to continue iteration.
2. **E next()**

- Description: Returns the next element in the iteration.
- Throws: NoSuchElementException if there are no more elements.
- Usage: Called within a loop if hasNext() return true.

```
Iterator<String> iterator = collection.iterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next());
}
```

3. **void remove()**

- Description: Removes the last element returned by the next() method from the underlying collection.
- Throws:
    - IllegalStateException if remove() is called without calling next() first.
    - UnsupportedOperationException if the remove operation is not supported by the collection.
- Usage: Allows removal of elements during iteration.

Key Characteristics of the Iterator Interface:

1. **Forward-Only Traversal**:
    - The Iterator interface supports only forward traversal of the collection. To traverse in reverse, use ListIterator.
2. **Safe Element Removal**:
    - Unlike modifying a collection directly during iteration (which may cause **ConcurrentModificationException**), the remove() method of Iterator allows safe removal.
3. **Universal Cursor**:
    - Iterator can be used with any collection in the Java Collections Framework, such as ArrayList, HashSet, LinkedList, etc. so it is also called universal cursor.
4. **Legacy vs Modern Cursors**:
    - Iterator replaced the older Enumeration interface and is more feature-rich.

3. List Interface Methods:

The **List** interface in Java is a part of the **java.util** package and extends the **Collection** interface. It represents an ordered collection of elements (often referred to as a sequence) that allows duplicate elements. Below is a list of its methods with descriptions:

Basic List Operations

1. **void add(int index, E element)**
   Inserts the specified element at the specified position in the list.

2. **boolean addAll(int index, Collection<? extends E> c)**
   Inserts all the elements from the specified collection into the list at the specified position.

3. **E get(int index)**
   Returns the element at the specified position in the list.

4. **E set(int index, E element)**
   Replaces the element at the specified position with the specified element and returns the old element.

5. **E remove(int index)**
   Removes the element at the specified position in the list and returns it.

6. **boolean remove(Object o)**
   Removes the first occurrence of the specified element from the list, if it is present.

7. **int indexOf(Object o)**
   Returns the index of the first occurrence of the specified element in the list, or -1 if the list does not contain it.
8. **int lastIndexOf(Object o)**
   Returns the index of the last occurrence of the specified element in the list, or -1 if the list does not contain it.

Iteration and Traversal:

9. **ListIterator<E> listIterator()**
   Returns a list iterator over the elements in the list (in proper sequence).
10. **ListIterator<E> listIterator(int index)**
    Returns a list iterator starting at the specified position in the list.
11. **Iterator<E> iterator()**
    Returns an iterator over the elements in the list.

Sublist View:

12. **List<E> subList(int fromIndex, int toIndex)**
    Returns a view of the portion of the list between the specified fromIndex (inclusive) and toIndex (exclusive).

4. ListIterator Interface Methods:

The **ListIterator** interface in Java, part of the **java.util** package, extends the **Iterator** interface and provides additional functionality for bidirectional traversal of a list. It is specifically designed to work with lists and offers operations to traverse the list both forward and backward, modify the list during iteration, and obtain the index of elements.

Methods of ListIterator Interface:

Traversal Methods

1. **boolean hasNext()**
   - Description: Checks if there is a next element in the list while traversing in the forward direction.
   - Returns: true if there is a next element, false otherwise.

2. **E next()**
   - Description: Returns the next element in the list and advances the cursor forward.
   - Throws: NoSuchElementException if no more elements exist in the forward direction.

3. **boolean hasPrevious()**
   - Description: Checks if there is a previous element in the list while traversing in the backward direction.
   - Returns: true if there is a previous element, false otherwise.

4. **E previous()**
    - Description: Returns the previous element in the list and moves the cursor backward.
    - Throws: NoSuchElementException if no more elements exist in the backward direction.

Index Methods:

5. **int nextIndex()**
    - **Description**: Returns the index of the element that would be returned by a subsequent call to next().
    - **Returns**: The index of the next element or the list size if the cursor is at the end of the list.

6. **int previousIndex()**
    - **Description**: Returns the index of the element that would be returned by a subsequent call to previous().
    - **Returns**: The index of the previous element or -1 if the cursor is at the beginning of the list.

Modification Methods:

7. **void remove()**
    - **Description**: Removes the last element returned by next() or previous() from the list.
    - **Throws**:
        - IllegalStateException if remove() is called without a preceding next() or previous() call.
        - UnsupportedOperationException if the remove operation is not supported by the list.

8. **void set(E e)**
    - **Description**: Replaces the last element returned by next() or previous() with the specified element.
    - **Throws**:
        - IllegalStateException if set() is called without a preceding next() or previous() call.
        - UnsupportedOperationException if the set operation is not supported by the list.

9. **void add(E e)**
    - **Description**: Inserts the specified element into the list at the current cursor position.
    - **Effect**: The new element is inserted before the element that would be returned by next() and after the element that would be returned by previous().
    - **Throws**:
        - UnsupportedOperationException if the add operation is not supported by the list.
        - ClassCastException or IllegalArgumentException if the specified element is not compatible with the list.

Key Characteristics of ListIterator:

1. **Bidirectional Traversal**:
    - Unlike Iterator, ListIterator allows traversal both forward (next()) and backward (previous()).

2. **Index Awareness**:
    - Provides methods to fetch the indices of the current position, which is particularly useful in list-based manipulations.

3. **Modification Support**:
    - Can add, update, and remove elements during iteration, making it a versatile tool for dynamic list operations.
4. **Works Only with Lists**:
    - Designed exclusively for lists like ArrayList, LinkedList, etc., and not with other collections.

5. ArrayList Methods:

The **ArrayList** class in Java, part of the **java.util** package, is a resizable array implementation of the **List** interface. It provides various methods for manipulating and accessing its elements. Below is a list of all the methods in the **ArrayList** class, grouped by their functionality, with detailed descriptions:

Methods of ArrayList are same as inherited from Collection and List:

1. Basic Operations

1. **boolean add(E e)**
    - **Description**: Appends the specified element to the end of the list.
    - **Returns**: true (as per the general contract of the Collection.add method).
2. **void add(int index, E element)**
    - **Description**: Inserts the specified element at the specified position in the list.
3. **boolean addAll(Collection<? extends E> c)**
    - **Description**: Appends all elements in the specified collection to the end of the list.
    - **Returns**: true if the list is modified.

4. **boolean addAll(int index, Collection<? extends E> c)**
    - **Description**: Inserts all elements from the specified collection starting at the specified position.
    - **Returns**: true if the list is modified.
5. **E get(int index)**
    - **Description**: Returns the element at the specified position in the list.
6. **E set(int index, E element)**
    - **Description**: Replaces the element at the specified position with the specified element.
    - **Returns**: The element previously at the specified position.
7. **E remove(int index)**
    - **Description**: Removes the element at the specified position in the list and returns it.
8. **boolean remove(Object o)**
    - **Description**: Removes the first occurrence of the specified element from the list, if present.
    - **Returns**: true if the list contained the element.
9. **void clear()**
    - **Description**: Removes all elements from the list.

---

2. Query Methods

10. **boolean contains(Object o)**
    - **Description**: Checks if the list contains the specified element.

- **Returns**: true if the list contains the element.

11. **boolean containsAll(Collection<?> c)**
    - **Description**: Checks if the list contains all elements in the specified collection.
    - **Returns**: true if the list contains all elements.
12. **int indexOf(Object o)**
    - **Description**: Returns the index of the first occurrence of the specified element, or -1 if not present.
13. **int lastIndexOf(Object o)**
    - **Description**: Returns the index of the last occurrence of the specified element, or -1 if not present.
14. **boolean isEmpty()**
    - **Description**: Checks if the list is empty.
    - **Returns**: true if the list contains no elements.
15. **int size()**
    - **Description**: Returns the number of elements in the list.

---

3. Bulk Operations

16. **boolean removeAll(Collection<?> c)**
    - **Description**: Removes all elements in the list that are also in the specified collection.
    - **Returns**: true if the list is modified.
17. **boolean retainAll(Collection<?> c)**
    - **Description**: Retains only the elements in the list that are also in the specified collection.

- **Returns**: true if the list is modified.

18. **Object[] toArray()**
    - **Description**: Returns an array containing all the elements in the list.
19. **<T> T[] toArray(T[] a)**
    - **Description**: Returns an array containing all the elements in the list in the specified array type.

---

4. Iteration and Views

20. **Iterator<E> iterator()**
    - **Description**: Returns an iterator over the elements in the list.
21. **ListIterator<E> listIterator()**
    - **Description**: Returns a list iterator over the elements in the list.
22. **ListIterator<E> listIterator(int index)**
    - **Description**: Returns a list iterator starting at the specified position.
23. **List<E> subList(int fromIndex, int toIndex)**
    - **Description**: Returns a view of the portion of the list between fromIndex (inclusive) and toIndex (exclusive).

---

5. Miscellaneous Methods

24. **void ensureCapacity(int minCapacity)**
    - **Description**: Increases the capacity of the list, if necessary, to ensure it can hold at least the number of elements specified by minCapacity.
25. **void trimToSize()**
    - **Description**: Trims the capacity of the list to be equal to its current size.

Default Methods (from Collection Interface)

26. **void forEach(Consumer<? super E> action)**
    - **Description**: Performs the given action for each element of the list.
27. **boolean removeIf(Predicate<? super E> filter)**
    - **Description**: Removes all elements of the list that satisfy the given predicate.
    - **Returns**: true if any elements were removed.
28. **void replaceAll(UnaryOperator<E> operator)**
    - **Description**: Replaces each element in the list with the result of applying the operator to that element.
29. **void sort(Comparator<? super E> c)**
    - **Description**: Sorts the elements of the list according to the specified comparator.

6. LinkedList Methods:

The **LinkedList** class in Java, part of the **java.util** package, implements the **List** and **Deque** interfaces. This allows it to function as a doubly-linked list with features of both a list and a queue. Below is a comprehensive list of all the methods in the **LinkedList** class, grouped by functionality, with detailed descriptions:

---

1. Basic Operations

1. **boolean add(E e)**
   - **Description**: Appends the specified element to the end of the list.
   - **Returns**: true (as per the Collection.add contract).
2. **void add(int index, E element)**
   - **Description**: Inserts the specified element at the specified position in the list.
3. **boolean addAll(Collection<? extends E> c)**
   - **Description**: Appends all elements in the specified collection to the end of the list.
   - **Returns**: true if the list is modified.
4. **boolean addAll(int index, Collection<? extends E> c)**
   - **Description**: Inserts all elements from the specified collection starting at the specified position.
   - **Returns**: true if the list is modified.
5. **E get(int index)**
   - **Description**: Returns the element at the specified position in the list.
6. **E set(int index, E element)**
   - **Description**: Replaces the element at the specified position with the specified element.

- **Returns**: The element previously at the specified position.

7. **E remove(int index)**
   - **Description**: Removes the element at the specified position in the list and returns it.
8. **boolean remove(Object o)**
   - **Description**: Removes the first occurrence of the specified element from the list.
   - **Returns**: true if the list contained the specified element.
9. **void clear()**
   - **Description**: Removes all elements from the list.

---

2. Query Methods

10. **boolean contains(Object o)**
    - **Description**: Checks if the list contains the specified element.
    - **Returns**: true if the list contains the element.
11. **boolean containsAll(Collection<?> c)**
    - **Description**: Checks if the list contains all elements in the specified collection.
    - **Returns**: true if the list contains all elements.
12. **int indexOf(Object o)**
    - **Description**: Returns the index of the first occurrence of the specified element, or -1 if not present.
13. **int lastIndexOf(Object o)**
    - **Description**: Returns the index of the last occurrence of the specified element, or -1 if not present.
14. **boolean isEmpty()**
    - **Description**: Checks if the list is empty.
    - **Returns**: true if the list contains no elements.
15. **int size()**
    - **Description**: Returns the number of elements in the list.

3. LinkedList-Specific Methods

Element Operations

16. **void addFirst(E e)**
    - **Description**: Inserts the specified element at the beginning of the list.
17. **void addLast(E e)**
    - **Description**: Appends the specified element to the end of the list.
18. **E getFirst()**
    - **Description**: Returns the first element in the list.
    - **Throws**: NoSuchElementException if the list is empty.
19. **E getLast()**
    - **Description**: Returns the last element in the list.
    - **Throws**: NoSuchElementException if the list is empty.
20. **E removeFirst()**
    - **Description**: Removes and returns the first element in the list.
    - **Throws**: NoSuchElementException if the list is empty.
21. **E removeLast()**
    - **Description**: Removes and returns the last element in the list.
    - **Throws**: NoSuchElementException if the list is empty.
22. **E poll()**
    - **Description**: Retrieves and removes the head of the list, or returns null if the list is empty.
23. **E pollFirst()**
    - **Description**: Retrieves and removes the first element of the list, or returns null if the list is empty.
24. **E pollLast()**
    - **Description**: Retrieves and removes the last element of the list, or returns null if the list is empty.

25. **E peek()**
    - **Description**: Retrieves the head of the list without removing it, or returns null if the list is empty.
26. **E peekFirst()**
    - **Description**: Retrieves the first element without removing it, or returns null if the list is empty.
27. **E peekLast()**
    - **Description**: Retrieves the last element without removing it, or returns null if the list is empty.

---

4. Iteration Methods

28. **Iterator<E> iterator()**
    - **Description**: Returns an iterator over the elements in the list.
29. **ListIterator<E> listIterator()**
    - **Description**: Returns a list iterator over the elements in the list.
30. **ListIterator<E> listIterator(int index)**
    - **Description**: Returns a list iterator starting at the specified position.
31. **DescendingIterator<E> descendingIterator()**
    - **Description**: Returns an iterator over the elements in reverse sequential order.

---

5. Queue/Deque Methods

32. **boolean offer(E e)**
    - **Description**: Adds the specified element as the tail of the list.
    - **Returns**: true if successful.

33. **boolean offerFirst(E e)**
    - **Description**: Inserts the specified element at the front of the list.
    - **Returns**: true if successful.
34. **boolean offerLast(E e)**
    - **Description**: Inserts the specified element at the end of the list.
    - **Returns**: true if successful.
35. **E element()**
    - **Description**: Retrieves, but does not remove, the head of the list.
    - **Throws**: NoSuchElementException if the list is empty.
36. **E remove()**
    - **Description**: Retrieves and removes the head of the list.
    - **Throws**: NoSuchElementException if the list is empty.

---

6. Bulk Operations

37. **boolean removeAll(Collection<?> c)**
    - **Description**: Removes all elements in the list that are also in the specified collection.
    - **Returns**: true if the list is modified.
38. **boolean retainAll(Collection<?> c)**
    - **Description**: Retains only the elements in the list that are also in the specified collection.
    - **Returns**: true if the list is modified.
39. **Object[] toArray()**
    - **Description**: Returns an array containing all elements in the list.
40. **<T> T[] toArray(T[] a)**
    - **Description**: Returns an array containing all elements in the list in the specified array type.

7. Vector Methods:

The **Vector** class in Java, part of the **java.util** package, is a legacy collection class that implements the **List** interface. It is synchronized and grows dynamically as elements are added. Below is a comprehensive list of all the methods in the **Vector** class, grouped by functionality, with detailed descriptions:

---

1. Basic Operations

1. **boolean add(E e)**
   - **Description**: Appends the specified element to the end of the vector.
   - **Returns**: true if the element is added.
2. **void add(int index, E element)**
   - **Description**: Inserts the specified element at the specified position in the vector.
3. **boolean addAll(Collection<? extends E> c)**
   - **Description**: Appends all elements in the specified collection to the end of the vector.
   - **Returns**: true if the vector is modified.
4. **boolean addAll(int index, Collection<? extends E> c)**
   - **Description**: Inserts all elements in the specified collection at the specified position in the vector.
   - **Returns**: true if the vector is modified.
5. **E get(int index)**
   - **Description**: Returns the element at the specified position in the vector.
6. **E set(int index, E element)**
   - **Description**: Replaces the element at the specified position with the specified element.
   - **Returns**: The element previously at the specified position.

7. **E remove(int index)**
    - **Description**: Removes the element at the specified position in the vector and returns it.
8. **boolean remove(Object o)**
    - **Description**: Removes the first occurrence of the specified element from the vector.
    - **Returns**: true if the vector contained the specified element.
9. **void clear()**
    - **Description**: Removes all elements from the vector.

---

2. Query Methods

10. **boolean contains(Object o)**
    - **Description**: Checks if the vector contains the specified element.
    - **Returns**: true if the vector contains the element.
11. **boolean containsAll(Collection<?> c)**
    - **Description**: Checks if the vector contains all elements in the specified collection.
    - **Returns**: true if the vector contains all elements.
12. **int indexOf(Object o)**
    - **Description**: Returns the index of the first occurrence of the specified element, or -1 if not present.
13. **int indexOf(Object o, int index)**
    - **Description**: Returns the index of the first occurrence of the specified element starting at the specified index, or -1 if not present.
14. **int lastIndexOf(Object o)**
    - **Description**: Returns the index of the last occurrence of the specified element, or -1 if not present.

15. **int lastIndexOf(Object o, int index)**
    - **Description**: Returns the index of the last occurrence of the specified element before or at the specified index, or -1 if not present.
16. **boolean isEmpty()**
    - **Description**: Checks if the vector is empty.
    - **Returns**: true if the vector contains no elements.
17. **int size()**
    - **Description**: Returns the number of elements in the vector.

---

3. Iteration Methods

18. **Iterator<E> iterator()**
    - **Description**: Returns an iterator over the elements in the vector.
19. **ListIterator<E> listIterator()**
    - **Description**: Returns a list iterator over the elements in the vector.
20. **ListIterator<E> listIterator(int index)**
    - **Description**: Returns a list iterator starting at the specified position in the vector.
21. **Enumeration<E> elements()**
    - **Description**: Returns an enumeration of the elements in the vector.

---

4. Bulk Operations

22. **boolean removeAll(Collection<?> c)**
    - **Description**: Removes all elements in the vector that are also in the specified collection.

- **Returns**: true if the vector is modified.

23. **boolean retainAll(Collection<?> c)**
    - **Description**: Retains only the elements in the vector that are also in the specified collection.
    - **Returns**: true if the vector is modified.
24. **Object[] toArray()**
    - **Description**: Returns an array containing all elements in the vector.
25. **<T> T[] toArray(T[] a)**
    - **Description**: Returns an array containing all elements in the vector in the specified array type.

---

5. Capacity and Synchronization

26. **void ensureCapacity(int minCapacity)**
    - **Description**: Ensures that the vector has at least the specified capacity.
27. **void trimToSize()**
    - **Description**: Trims the capacity of the vector to its current size.
28. **int capacity()**
    - **Description**: Returns the current capacity of the vector.

---

6. Stack-Like Methods

29. **void addElement(E obj)**
    - **Description**: Adds the specified element to the end of the vector.
30. **E firstElement()**
    - **Description**: Returns the first element of the vector.
    - **Throws**: NoSuchElementException if the vector is empty.

31. **E lastElement()**
    - **Description**: Returns the last element of the vector.
    - **Throws**: NoSuchElementException if the vector is empty.
32. **void insertElementAt(E obj, int index)**
    - **Description**: Inserts the specified element at the specified position in the vector.
33. **void removeElementAt(int index)**
    - **Description**: Removes the element at the specified position in the vector.
34. **boolean removeElement(Object obj)**
    - **Description**: Removes the first occurrence of the specified element.
    - **Returns**: true if the vector contained the specified element.
35. **void setElementAt(E obj, int index)**
    - **Description**: Sets the element at the specified position to the specified object.
36. **void copyInto(Object[] anArray)**
    - **Description**: Copies the elements of the vector into the specified array.

---

7. Miscellaneous Methods

37. **void forEach(Consumer<? super E> action)**
    - **Description**: Performs the given action for each element of the vector.
38. **boolean removeIf(Predicate<? super E> filter)**
    - **Description**: Removes all elements that satisfy the given predicate.
    - **Returns**: true if any elements were removed.
39. **void replaceAll(UnaryOperator<E> operator)**
    - **Description**: Replaces each element of the vector with the result of applying the operator.

40. **void sort(Comparator<? super E> c)**
    - **Description**: Sorts the elements of the vector according to the specified comparator.

8. Stack Methods:

The **Stack** class in Java, part of the **java.util** package, is a subclass of **Vector** that provides a last-in-first-out (LIFO) stack data structure. It has specific methods for stack operations in addition to inheriting all methods from the **Vector** class.

Here's a comprehensive list of all the methods in the **Stack** class:

---

1. Stack-Specific Methods

1. **E push(E item)**
    - **Description**: Pushes the specified item onto the top of the stack.
    - **Returns**: The item pushed.
2. **E pop()**
    - **Description**: Removes and returns the item at the top of the stack.
    - **Throws**: EmptyStackException if the stack is empty.
3. **E peek()**
    - **Description**: Returns the item at the top of the stack without removing it.
    - **Throws**: EmptyStackException if the stack is empty.

4. **boolean empty()**
    - **Description**: Checks if the stack is empty.
    - **Returns**: true if the stack contains no items.
5. **int search(Object o)**
    - **Description**: Returns the 1-based position of the specified object in the stack, counting from the top of the stack.
    - **Returns**: The position of the object, or -1 if the object is not found.

---

2. Inherited Methods from Vector

Since **Stack** extends **Vector**, it inherits all its methods. Below are some commonly used methods:

Basic Operations

6. **boolean add(E e)**
    - **Description**: Adds the specified element to the end of the vector.
7. **void addElement(E obj)**
    - **Description**: Adds the specified object to the end of the vector.
8. **E remove(int index)**
    - **Description**: Removes the element at the specified position in the vector.
9. **boolean remove(Object o)**
    - **Description**: Removes the first occurrence of the specified element from the vector.

10. **void clear()**
    - **Description**: Removes all elements from the vector.

Query Operations

11. **int size()**
    - **Description**: Returns the number of elements in the stack.
12. **boolean isEmpty()**
    - **Description**: Checks if the vector (or stack) is empty.
13. **boolean contains(Object o)**
    - **Description**: Checks if the vector contains the specified element.
14. **int indexOf(Object o)**
    - **Description**: Returns the index of the first occurrence of the specified element.
15. **E elementAt(int index)**
    - **Description**: Returns the element at the specified position in the vector.

Iteration

16. **Enumeration<E> elements()**
    - **Description**: Returns an enumeration of the elements in the vector.
17. **Iterator<E> iterator()**
    - **Description**: Returns an iterator over the elements in the vector.

Capacity Operations

18. **int capacity()**
    - **Description**: Returns the current capacity of the vector.
19. **void ensureCapacity(int minCapacity)**
    - **Description**: Ensures that the vector has at least the specified capacity.
20. **void trimToSize()**
    - **Description**: Trims the capacity of the vector to match its current size.

Notes:

- **Thread Safety**: The **Stack** class is synchronized because it inherits from **Vector**, but it is recommended to use **Deque** (like **ArrayDeque**) for a more modern and efficient stack implementation in single-threaded contexts.
- **Legacy Class**: While **Stack** is still widely used, newer applications often prefer alternatives such as **ArrayDeque** for stack operations.

==========END==========