Set Interface Methods:-

Shambhu Kumar

@Kumarsam07

@javac_java

http://linkedin.com/in/kumarsam07

1. Set Interface Methods:

The Collection interface in Java, part of the java.util package, defines the The Set interface in Java, part of the java.util package, extends the Collection interface and represents a collection of unique elements. Since Set is an interface, it cannot be instantiated directly. Common implementations include HashSet, LinkedHashSet, and TreeSet.

Here is a complete list of methods in the Set interface, which are inherited from the Collection interface:

---

1. Basic Operations

1. **boolean add(E e):**
   - Description: Adds the specified element to the set if it is not already present.
   - Returns: true if the element was added; false if the element already exists.
2. **boolean addAll(Collection<? extends E> c)**
   - Description: Adds all elements from the specified collection to the set, ignoring duplicates.
   - Returns: true if the set was modified.

3. **void clear()**
    - Description: Removes all elements from the set.
4. **boolean remove(Object o)**
    - Description: Removes the specified element from the set if it exists.
    - Returns: true if the element was removed.
5. **boolean removeAll(Collection<?> c)**
    - Description: Removes all elements in the set that are also contained in the specified collection.
    - Returns: true if the set was modified.
6. **boolean retainAll(Collection<?> c)**
    - Description: Retains only the elements in the set that are also contained in the specified collection.
    - Returns: true if the set was modified.

---

2. Query Operations

7. **boolean contains(Object o)**
    - Description: Checks if the set contains the specified element.
    - Returns: true if the element exists.
8. **boolean containsAll(Collection<?> c)**
    - Description: Checks if the set contains all elements in the specified collection.
    - Returns: true if all elements are present.

9. **boolean isEmpty()**
    - Description: Checks if the set is empty.
    - Returns: true if the set has no elements.
10. **int size()**
    - Description: Returns the number of elements in the set.

---

3. Iteration and Streams

11. **Iterator<E> iterator()**
    - Description: Returns an iterator over the elements in the set.
12. **void forEach(Consumer<? super E> action)**
    - Description: Performs the specified action for each element in the set.
13. **Spliterator<E> spliterator()**
    - Description: Returns a spliterator over the elements in the set for parallel processing.
14. **Stream<E> stream()**
    - Description: Returns a sequential stream with the set as its source.
15. **Stream<E> parallelStream()**
    - Description: Returns a parallel stream with the set as its source.

---

4. Bulk Operations

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

Implementation Notes

- **HashSet:** Backed by a HashMap, does not guarantee the order of elements.
- **LinkedHashSet:** Maintains insertion order.
- **TreeSet:** Maintains elements in sorted (natural or custom comparator) order.

2. HashSet Methods:

The HashSet class in Java is part of the java.util package and implements the Set interface. It uses a hash table for storing unique elements, ensuring no duplicates.

---

1. HashSet-Specific Constructors:

1. **HashSet():**
    - Description: Constructs an empty HashSet with the default initial capacity (16) and load factor (0.75).

2. **HashSet(int initialCapacity)**
    - Description: Constructs an empty HashSet with the specified initial capacity and default load factor.
3. **HashSet(int initialCapacity, float loadFactor)**
    - Description: Constructs an empty HashSet with the specified initial capacity and load factor.
4. **HashSet(Collection<? extends E> c)**
    - Description: Constructs a new HashSet containing the elements of the specified collection.

Methods are same Inherited from the Set Interface.

Key Characteristics of HashSet

- **No Duplicates:** Ensures that each element is unique.
- **Order:** Does not guarantee any specific order of elements.
- **Null Elements**: Allows a single null element.
- **Performance:** Provides constant-time performance for basic operations like add, remove, and contains (on average).

3. LinkedHashSet Methods:

The **LinkedHashSet** class in Java is part of the **java.util** package. It extends the **HashSet** class and implements the **Set** interface. **LinkedHashSet** differs from **HashSet** by maintaining the **insertion order** of elements, making it useful when order is important.

Constructors of LinkedHashSet

1. **LinkedHashSet()**
    - **Description**: Creates an empty LinkedHashSet with the default initial capacity (16) and load factor (0.75).

2. **LinkedHashSet(int initialCapacity)**
    - **Description**: Creates a LinkedHashSet with the specified initial capacity and default load factor (0.75).
3. **LinkedHashSet(int initialCapacity, float loadFactor)**
    - **Description**: Creates a LinkedHashSet with the specified initial capacity and load factor.
4. **LinkedHashSet(Collection<? extends E> c)**
    - **Description**: Creates a LinkedHashSet containing all elements from the specified collection, maintaining the insertion order.

Methods are same Inherited from the Set Interface.

Characteristics of LinkedHashSet

- **Insertion Order**: Maintains the order in which elements are added.
- **No Duplicates**: Ensures unique elements, just like **HashSet**.
- **Performance**: Slightly slower than **HashSet** for operations due to the overhead of maintaining insertion order.
- **Null Elements**: Allows a single null element.

4. TreeSet Methods:

The **TreeSet** class in Java is part of the **java.util** package and implements the **NavigableSet** interface, which in turn extends **SortedSet**. **TreeSet** is a collection that is sorted in ascending order by default (using natural ordering or a custom comparator). It is backed by a **TreeMap**, and it does not allow duplicate elements.

Constructors

1. **TreeSet()**
   Creates an empty TreeSet that orders its elements according to their natural ordering.
2. **TreeSet(Collection<? extends E> c)**
   Constructs a TreeSet containing the elements of the specified collection, ordered according to the natural ordering of its elements.

Methods

Basic Operations

1. **boolean add(E e)**
   Adds the specified element to the set if it is not already present.
2. **boolean remove(Object o)**
   Removes the specified element from the set if it is present.
3. **void clear()**
   Removes all elements from the set.
4. **int size()**
   Returns the number of elements in the set.
5. **boolean isEmpty()**
   Returns true if the set contains no elements.
6. **boolean contains(Object o)**
   Returns true if the set contains the specified element.

Navigational Methods

1. **E first()**
   Returns the first (lowest) element in the set.
2. **E last()**
   Returns the last (highest) element in the set.
3. **E lower(E e)**
   Returns the greatest element in this set strictly less than the given element, or null if no such element exists.
4. **E higher(E e)**
   Returns the smallest element in this set strictly greater than the given element, or null if no such element exists.
5. **E floor(E e)**
   Returns the greatest element in this set less than or equal to the given element, or null if no such element exists.
6. **E ceiling(E e)**
   Returns the smallest element in this set greater than or equal to the given element, or null if no such element exists.

Set Views

1. **SortedSet<E> headSet(E toElement)**
   Returns a view of the portion of this set whose elements are strictly less than the given element.
2. **SortedSet<E> headSet(E toElement, boolean inclusive)** *(Java 1.6+)*
   Returns a view of the portion of this set whose elements are less than (or equal to, if inclusive is true) the given element.
3. **SortedSet<E> tailSet(E fromElement)**
   Returns a view of the portion of this set whose elements are greater than or equal to the given element.

4. **SortedSet<E> tailSet(E fromElement, boolean inclusive)** *(Java 1.6+)*
   Returns a view of the portion of this set whose elements are greater than (or equal to, if inclusive is true) the given element.
5. **SortedSet<E> subSet(E fromElement, E toElement)**
   Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive.
6. **SortedSet<E> subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)** *(Java 1.6+)*
   Returns a view of the portion of this set whose elements range from fromElement to toElement, with control over inclusiveness.

Iterator

1. **Iterator<E> iterator()**
   Returns an iterator over the elements in this set in ascending order.
2. **Iterator<E> descendingIterator()**
   Returns an iterator over the elements in this set in descending order.

Other Methods

1. **Comparator<? super E> comparator()**
   Returns the comparator used to order the elements in this set, or null if it uses the natural ordering.
2. **Object clone()**
   Returns a shallow copy of this TreeSet.
3. **boolean equals(Object o)**
   Compares the specified object with this set for equality.

4. **int hashCode()**
   Returns the hash code value for this set.
5. **Spliterator<E> spliterator()** *(Java 8+)*
   Creates a Spliterator over the elements in this set.

Notes

- The TreeSet is implemented using a **Red-Black Tree**, ensuring that elements are always sorted and that operations such as add, remove, and contains have a time complexity of O(logn).
- The class does not allow null elements if the natural ordering or a comparator is used.
- TreeSet maintains elements in natural or custom order defined by a comparator.
- It does **not** allow duplicate elements.
- All methods have logarithmic time complexity (O(log n)).

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