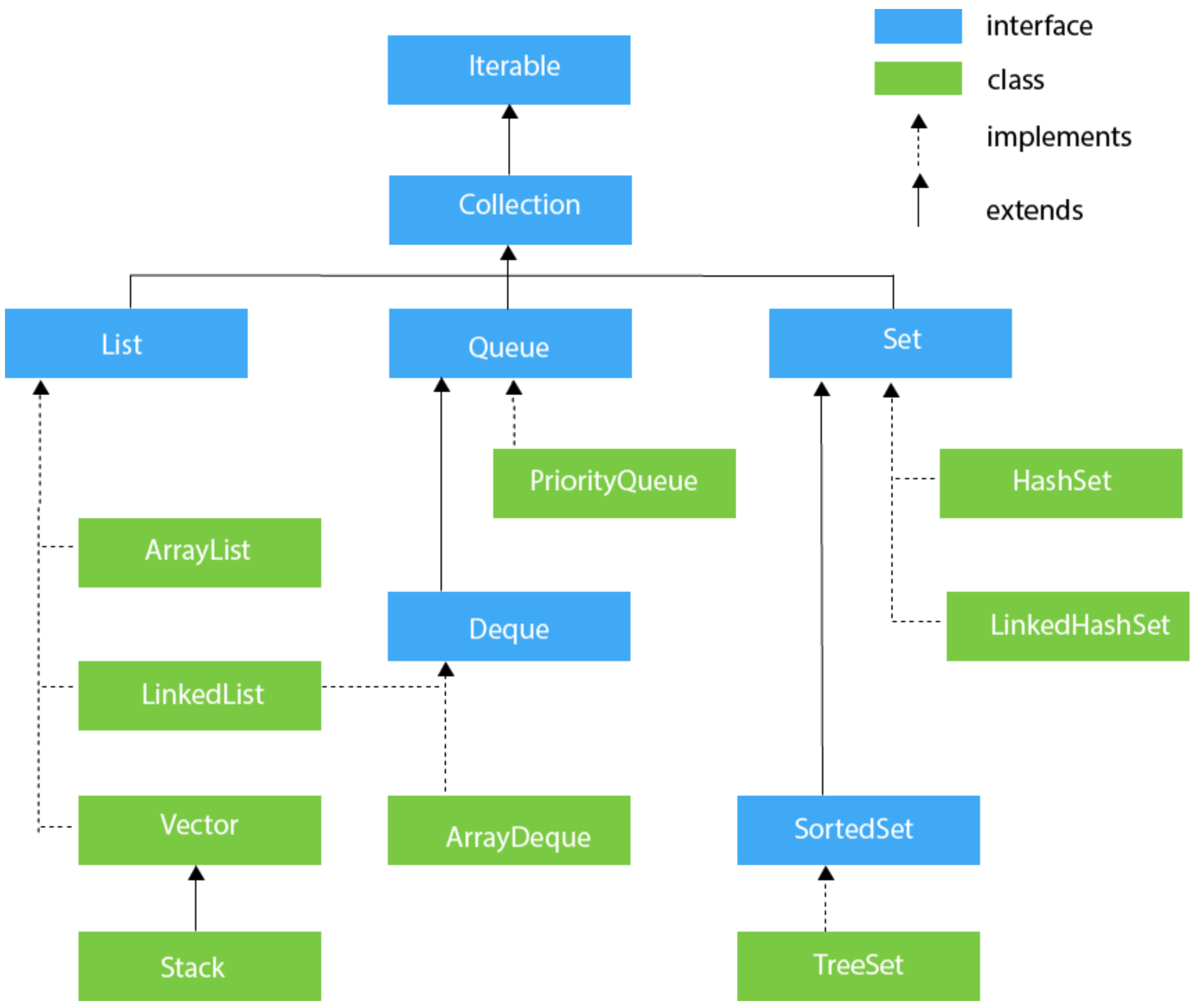
13.3 - Set Interface in Java

👉 Introduction

In Java, one of the most commonly used interfaces in the Collection framework is the List interface, which allows duplicate elements and supports operations like adding, sorting, and inserting and fetching elements based on index. Two popular classes implementing the List interface are ArrayList and LinkedList. These classes allow you to add duplicate elements and maintain the order in which elements were inserted.



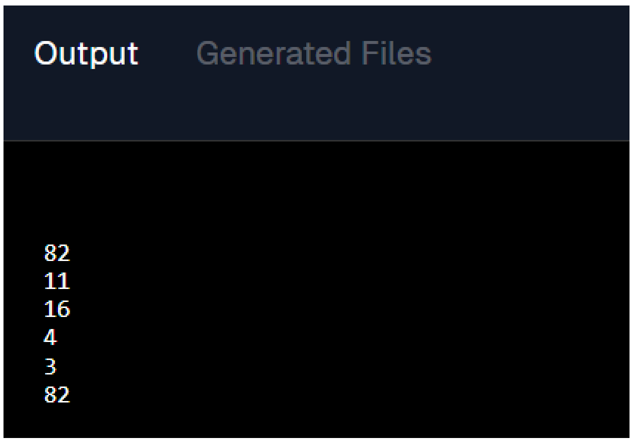
👉Example of Using a List

```
import java.util.*;

public class Demo {
    public static void main(String[] args) {
        List<Integer> nums = new ArrayList<Integer>();
        nums.add(82);
        nums.add(11);
        nums.add(16);
        nums.add(4);
        nums.add(3);
        nums.add(82); // Duplicate element

        for (Object num : nums) {
            System.out.println(num);
        }
    }
}
```

👉Output:



```
82
11
16
4
3
82
```

In the above example, duplicate elements (82) are allowed. However, if you need to store unique values where duplicates are not permitted, the Set interface is the appropriate choice.

👉 Set Interface

The Set interface is a part of the Collection framework in Java, designed to hold a collection of unique elements but it also doesn't supports index based operations as List does . It does not allow duplicate values and provides functionalities to ensure all elements in the collection are distinct.

👉 Characteristics of Set Interface:

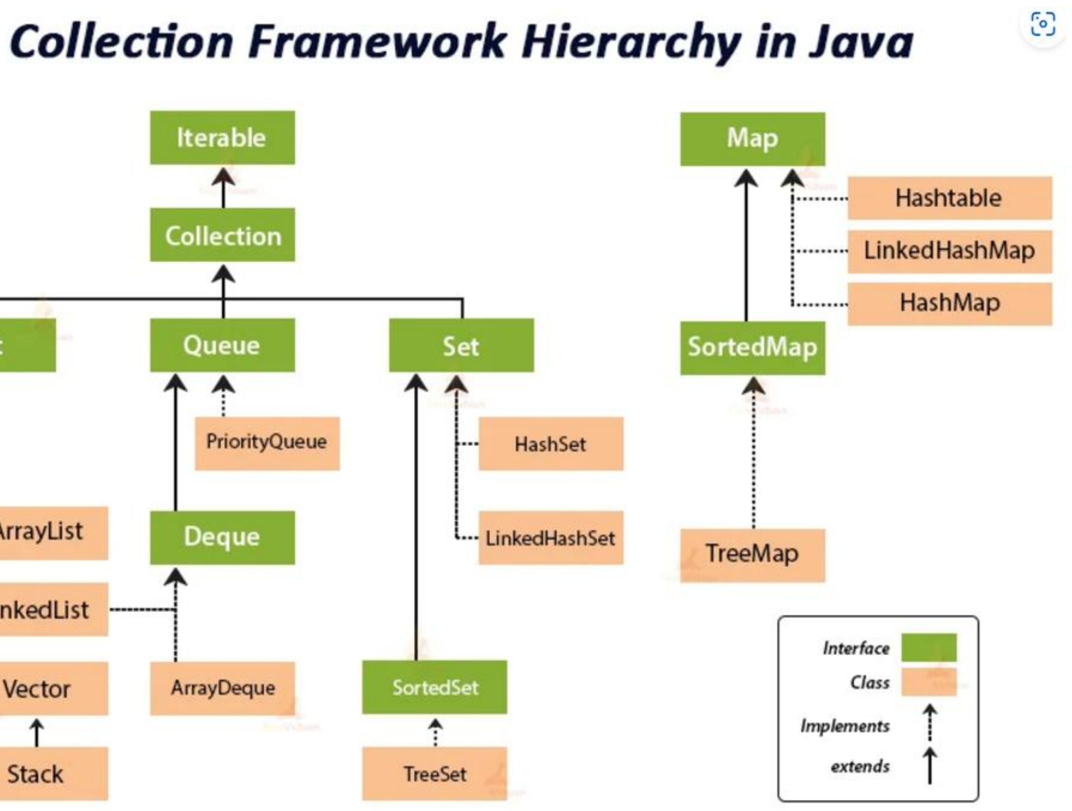
- Extends the Collection interface.
- Guarantees that no duplicate elements are present.
- Implemented by classes like HashSet, LinkedHashSet(For index based operations), and TreeSet(For ordered Sorting of elements).

👉 Syntax for Declaring a Set

Set<Integer> values = new HashSet<Integer>();

👉 Implementing Classes of Set Interface

1. **HashSet**:
    - Does not maintain any order of elements.
    - Uses a hashing algorithm for storing elements, making retrieval fast.
2. **LinkedHashSet**:
    - Maintains the insertion order of elements.
    - Slower than HashSet due to the maintenance of the order.
3. **TreeSet**:
    - Stores elements in sorted order (ascending).
    - Implements the NavigableSet interface, which extends the SortedSet interface.
    - Ensures that elements are both unique and sorted.



👉 Methods in Set Interface

1. add()

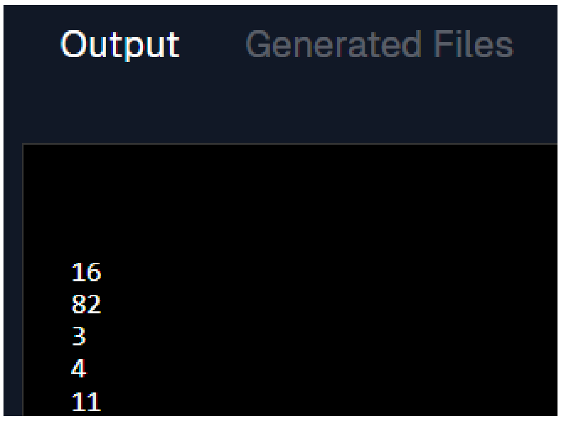
The add() method is used to insert elements into the set. If an element already exists in the set, it will not be added again (no duplicates are allowed).

👉 Example: Using HashSet

```
import java.util.*;

public class Demo {
    public static void main(String[] args) {
        Set<Integer> values = new HashSet<Integer>();
        values.add(82);
        values.add(11);
        values.add(16);
        values.add(4);
        values.add(3);
        values.add(82); // Duplicate element

        for (int num : values) {
            System.out.println(num);
            }
        }
}
```



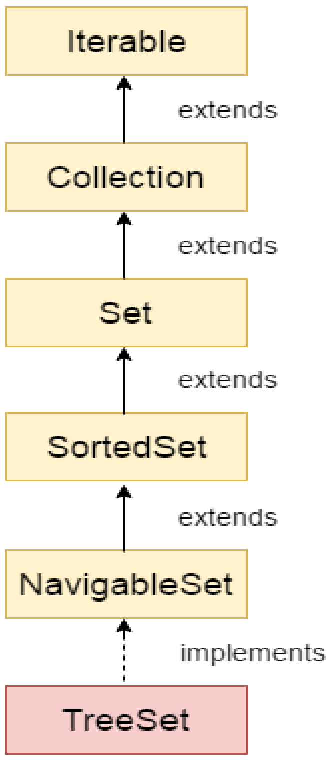
```
16
82
3
4
11
```

👉**Explanation**:

- The output may not appear in the order in which the elements were added. This is because HashSet does not maintain insertion order.
- Duplicate values (82) are removed automatically as Set Supports only Unique values.

👉Using TreeSet for Sorted and Unique Values

If you need a collection where the elements are both unique and sorted, you should use the TreeSet class. The TreeSet class implements the NavigableSet interface, which extends the SortedSet interface. As a result, elements are stored in a sorted manner.



👉Example: Using TreeSet

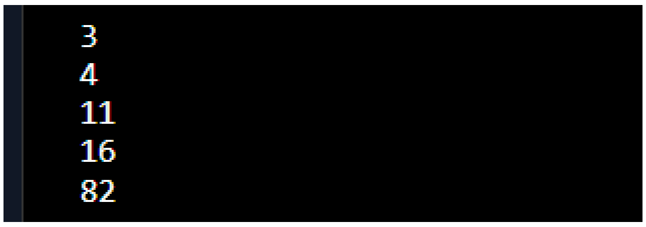
```
import java.util.*;

public class Demo {
    public static void main(String[] args) {
        Set<Integer> values = new
TreeSet<Integer>();
```

```
        values.add(82);
        values.add(11);
        values.add(16);
        values.add(4);
        values.add(3);
        values.add(82); // Duplicate element

        for (int num : values) {
            System.out.println(num);
        }
    }
}
```

👉**Output**:



```
3
4
11
16
82
```

- The output is sorted in ascending order because of the TreeSet implementation.
- Duplicate values are not added.

👉Understanding the Iterable Interface

The Iterable interface is the top-most interface in the Collection framework, which allows for iterating over a collection. It provides the ability to traverse the elements using an iterator.

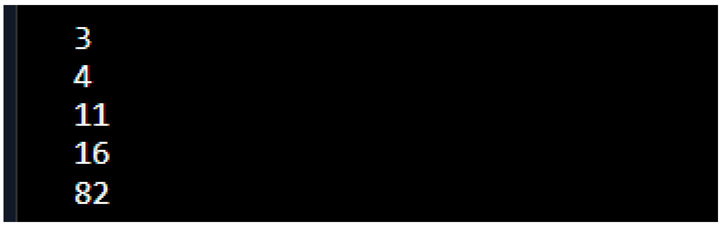
👉Example: Using Iterator with TreeSet

```
import java.util.*;

public class Demo {
    public static void main(String[] args) {
        Set<Integer> values = new TreeSet<Integer>();
        values.add(82);
        values.add(11);
        values.add(16);
        values.add(4);
        values.add(3);
        values.add(82); // Duplicate element

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

👉 Output



```
3
4
11
16
82
```

👉 Methods in Iterator

1. **hasNext()**: Checks if there are more elements in the collection.
2. **next()**: Retrieves the next element in the collection.