HowToDoInJava

Java LinkedHashSet class

Last Updated: December 26, 2020 By: Lokesh Gupta Java Collections Java HashSet

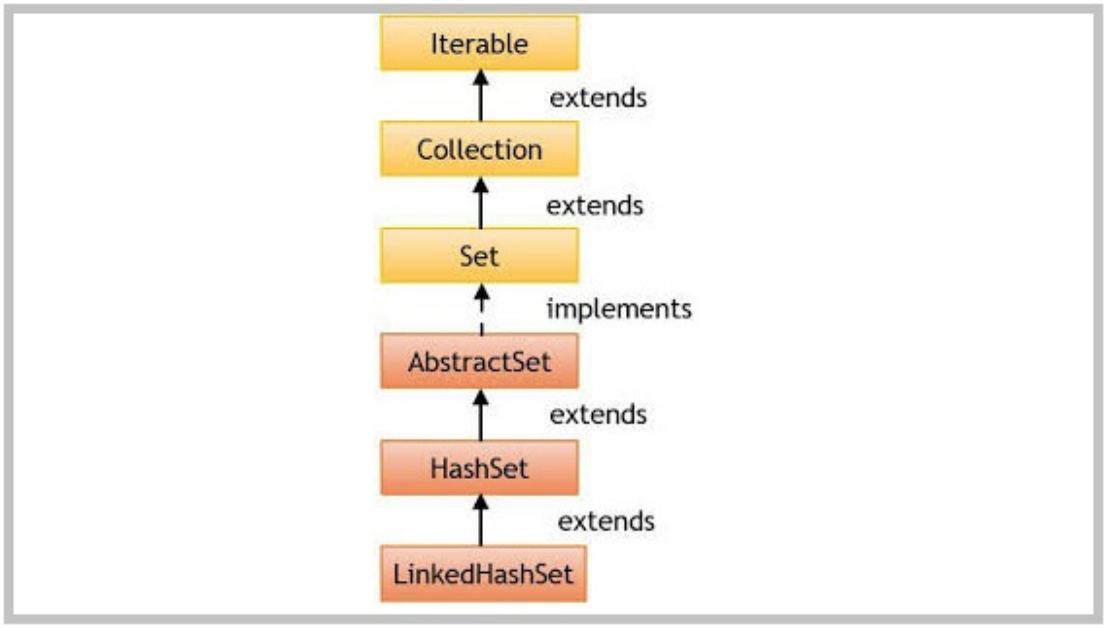
Java LinkedHashSet class **extends HashSet** and `implements Set` interface. It is very very similar to HashSet class, except if offers the ***predictable iteration order***.

Table of Contents

1. LinkedHashSet Hierarchy

The LinkedHashSet class extends `HashSet` class and implements `Set` interface. The Set interface inherits `Collection` and `Iterable` interfaces in hierarchical order.

```
public class LinkedHashSet<E> extends HashSet<E>
        implements Set<E>, Cloneable, Serializable
{
  //implementation
}
```



LinkedHashSet Hierarchy

2. LinkedHashSet Features

- It extends `HashSet` class which extends `AbstractSet` class.
- It implements `Set` interface.
- **Duplicate values are not allowed** in LinkedHashSet.
- One NULL element is allowed in LinkedHashSet.
- It is an **ordered collection** which is the order in which elements were inserted into the set (**insertion-order**).
- Like HashSet, this class offers **constant time performance** for the basic operations(add, remove, contains and size).
- LinkedHashSet is **not synchronized**. If multiple threads access a hash set concurrently, and at least one of the threads modifies the set, it must be synchronized externally.
- Use **Collections.synchronizedSet(new LinkedHashSet())** method to get the synchronized LinkedHashSet.
- The iterators returned by this class's iterator method are **fail-fast** and may throw `ConcurrentModificationException` if the set is modified at any time after the iterator is created, in any way except through the iterator's own `remove()` method.

- LinkedHashSet also implements Searlizable and Cloneable interfaces.

2.1. Initial Capacity

The initial capacity means the number of buckets (in backing HashMap) when LinkedHashSet is created. The number of buckets will be automatically increased if the current size gets full.

Default initial capacity is **16**. We can override this default capacity by passing default capacity in it's constructor **LinkedHashSet(int initialCapacity)**.

2.2. Load Factor

The load factor is a measure of how full the LinkedHashSet is allowed to get before its capacity is automatically increased. Default load factor is **0.75**.

This is called **threshold** and is equal to (DEFAULT_LOAD_FACTOR * DEFAULT_INITIAL_CAPACITY). When LinkedHashSet elements count exceed this threshold, LinkedHashSet is resized and new capacity is double the previous capacity.

With default LinkedHashSet, the internal capacity is 16 and load factor is 0.75. The number of buckets will automatically get increased when the table has 12 elements in it.

3. LinkedHashSet Constructors

The LinkedHashSet has four types of constructors:

1. **LinkedHashSet():** initializes a default LinkedHashSet instance with the default initial capacity (16) and load factor (0.75).
2. **LinkedHashSet(int capacity):** initializes a LinkedHashSet with a specified capacity and load factor (0.75).
3. **LinkedHashSet(int capacity, float loadFactor):** initializes LinkedHashSet with specified initial capacity and load factor.

4. **LinkedHashSet(Collection c):** initializes a LinkedHashSet with same elements as the specified collection.

4. LinkedHashSet Methods

1. **public boolean add(E e)** : adds the specified element to the Set if not already present. This method internally uses **equals()** method to check for duplicates. If element is duplicate then element is rejected and value is NOT replaced.
2. **public void clear()** : removes all the elements from the LinkedHashSet.
3. **public boolean contains(Object o)** : returns `true` if the LinkedHashSet contains the specified element, othrweise `false`.
4. **public boolean isEmpty()** : returns `true` if LinkedHashSet contains no element, otherwise `false`.
5. **public int size()** : returns the number of elements in the LinkedHashSet.
6. **public Iterator<E> iterator()** : returns an iterator over the elements in this LinkedHashSet. The elements are returned from iterator in no specific order.
7. **public boolean remove(Object o)** : removes the specified element from the LinkedHashSet if it is present and return `true`, else returns `false`.
8. **public boolean removeAll(Collection<?> c)** : remove all the elements in the LinkedHashSet that are part of the specified collection.
9. **public Object clone()** : returns a shallow copy of the LinkedHashSet.
10. **public Spliterator<E> spliterator()** : creates a late-binding and fail-fast Spliterator over the elements in this LinkedHashSet. It has following intialization properties `Spliterator.DISTINCT`, `Spliterator.ORDERED`.

5. LinkedHashSet Example

5.1. LinkedHashSet add, remove, iterator example

```
Java LinkedHashSet Example
```

```
//1. Create LinkedHashSet
LinkedHashSet<String> LinkedHashSet = new LinkedHashSet<>();

//2. Add elements to LinkedHashSet
LinkedHashSet.add("A");
LinkedHashSet.add("B");
LinkedHashSet.add("C");
LinkedHashSet.add("D");
LinkedHashSet.add("E");

System.out.println(LinkedHashSet);

//3. Check if element exists
boolean found = LinkedHashSet.contains("A");        //true
System.out.println(found);

//4. Remove an element
LinkedHashSet.remove("D");

//5. Iterate over values
Iterator<String> itr = LinkedHashSet.iterator();

while(itr.hasNext())
{
    String value = itr.next();

    System.out.println("Value: " + value);
}
```

Program Output.

```
Console

[A, B, C, D, E]
true
Value: A
Value: B
Value: C
Value: E
```

5.2. Convert LinkedHashSet to Array Example

Java example to convert a LinkedHashSet to array using **toArrray()** method.

```
Java LinkedHashSet Example
```

```
LinkedHashSet<String> LinkedHashSet = new LinkedHashSet<>();

LinkedHashSet.add("A");
LinkedHashSet.add("B");
LinkedHashSet.add("C");
LinkedHashSet.add("D");
LinkedHashSet.add("E");

String[] values = new String[LinkedHashSet.size()];

LinkedHashSet.toArray(values);

System.out.println(Arrays.toString(values));
```

Program Output.

```
Console

[A, B, C, D, E]
```

5.3. Convert LinkedHashSet to ArrayList Example

Java example to convert a LinkedHashSet to arraylist using *Java 8 stream API*.

```
Java LinkedHashSet Example

LinkedHashSet<String> LinkedHashSet = new LinkedHashSet<>();

LinkedHashSet.add("A");
LinkedHashSet.add("B");
LinkedHashSet.add("C");
LinkedHashSet.add("D");
LinkedHashSet.add("E");

List<String> valuesList = LinkedHashSet.stream().collect(Collectors.toList());

System.out.println(valuesList);
```

Program Output.

```
Console

[A, B, C, D, E]
```

6. LinkedHashSet Usecases

LinkedHashSet is very much like ArrayList (ordered) and HashSet (unique elements). It additionally guarantees the iteration order of elements (in order elements were inserted).

A real life usecase for LinkedHashSet can be storing data from stream where stream may contain duplicate records in the desired order, and we are only interested in distinct records but in exactly same order.

Another usecase can be finding distinct words in a given sentence and order of words should be fixed as they apprear in the senetence.

7. LinkedHashSet Performance

- LinkedHashSet class offers **constant time performance of O(1)**for the basic operations(add, remove, contains and size), assuming the hash function disperses the elements properly among the buckets.
- Performance is likely to be just slightly below that of HashSet, due to the added expense of maintaining the linked list, with one exception of iteration.
  Iteration over a LinkedHashSet requires time proportional to the size of the set, regardless of its capacity. Iteration over a HashSet is likely to be more expensive, requiring time proportional to its capacity. Thus LinkedHashSet may provide better performance than HashSet while iteration.

8. Conclusion

From above discussion, it is evident that LinkedHashSet is very useful collection class in cases where we want to handle duplicate records in some fixed order. It provided predictable performance for basic operations.

If iteration order of elements is not needed then it is recommended to use the lighter-weight HashSet and HashMap instead.

Drop me your questions related to **LinkedHashSet in Java** in comments.

Happy Learning !!

Reference:

LinkedHashSet Java Docs

Recommended Reading:

1. [Solved]: javax.xml.bind.JAXBException: class java.util.ArrayList nor any of its super class is known to this context
2. Java HashSet class
3. Java CopyOnWriteArraySet class
4. Java TransferQueue – Java LinkedTransferQueue class
5. Java LinkedHashMap class
6. Java TreeMap class
7. Java Hashtable class
8. Java PriorityBlockingQueue class
9. Java ArrayBlockingQueue class
0. Java CopyOnWriteArrayList class

Leave a Comment

Name *

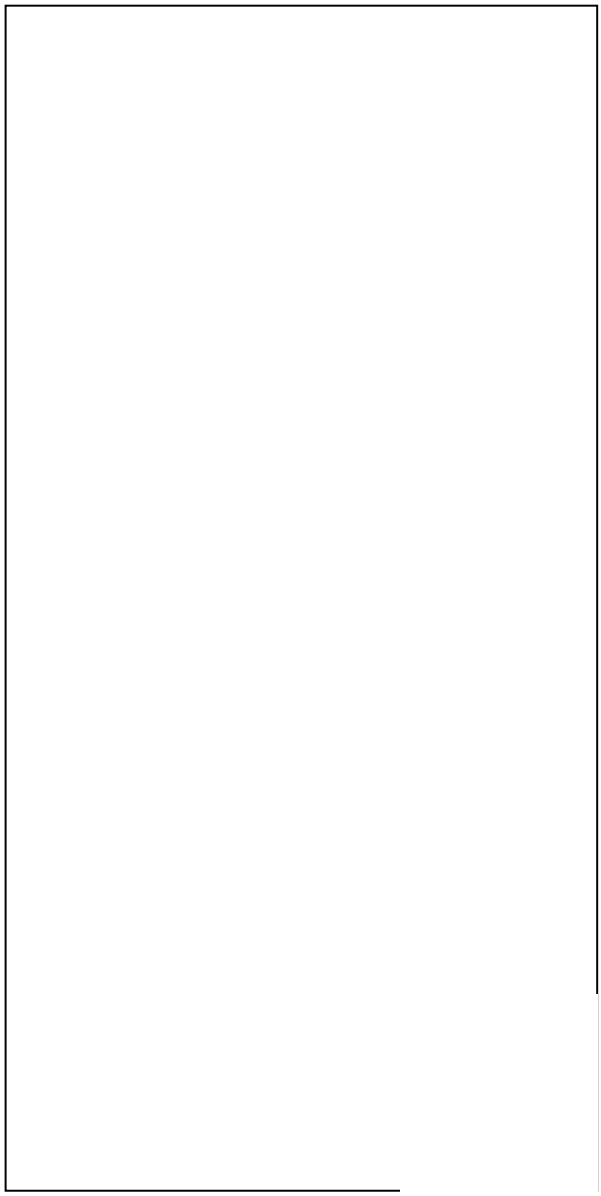
Email *

Website

Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search ...




VIDEOGAMER
NEW SOUTH PARK GAME
WITHOUT UBISOFT COMING?!

HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise

Guest Posts

Blogs

REST API Tutorial