java.util

Class LinkedHashSet<E>

java.lang.Object
  java.util.AbstractCollection<E>
    java.util.AbstractSet<E>
      java.util.HashSet<E>
        java.util.LinkedHashSet<E>

Type Parameters:

`E` - the type of elements maintained by this set

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, Set<E>

```
public class LinkedHashSet<E>
extends HashSet<E>
implements Set<E>, Cloneable, Serializable
```

Hash table and linked list implementation of the `Set` interface, with predictable iteration order. This implementation differs from `HashSet` in that it maintains a doubly-linked list running through all of its entries. This linked list defines the iteration ordering, which is the order in which elements were inserted into the set (*insertion-order*). Note that insertion order is *not* affected if an element is *re-inserted* into the set. (An element `e` is reinserted into a set `s` if `s.add(e)` is invoked when `s.contains(e)` would return `true` immediately prior to the invocation.)

This implementation spares its clients from the unspecified, generally chaotic ordering provided by `HashSet`, without incurring the increased cost associated with `TreeSet`. It can be used to produce a copy of a set that has the same order as the original, regardless of the original set's implementation:

```
void foo(Set s) {
    Set copy = new LinkedHashSet(s);
    ...
}
```

This technique is particularly useful if a module takes a set on input, copies it, and later returns results whose order is determined by that of the copy. (Clients generally appreciate having things returned in the same order they were presented.)

This class provides all of the optional `Set` operations, and permits null elements. Like `HashSet`, it provides constant-time performance for the basic operations (`add`, `contains` and `remove`), assuming the hash function disperses 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: 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*.

A linked hash set has two parameters that affect its performance: *initial capacity* and *load factor*. They are defined precisely as for `HashSet`. Note, however, that the penalty for choosing an excessively high value for initial capacity is less severe for this class than for `HashSet`, as iteration times for this class are unaffected by capacity.

Note that this implementation is not synchronized. If multiple threads access a linked hash set concurrently, and at least one of the threads modifies the set, it *must* be synchronized externally. This is typically accomplished by synchronizing on some object that naturally encapsulates the set. If no such object exists, the set should be "wrapped" using the `Collections.synchronizedSet` method. This is best done at creation time, to prevent accidental unsynchronized access to the set:

```
Set s = Collections.synchronizedSet(new LinkedHashSet(...));
```

The iterators returned by this class's `iterator` method are *fail-fast*: if the set is modified at any time after the iterator is created, in any way except through the iterator's own `remove` method, the iterator will throw a `ConcurrentModificationException`. Thus, in

the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Note that the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw `ConcurrentModificationException` on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: *the fail-fast behavior of iterators should be used only to detect bugs.*

This class is a member of the Java Collections Framework.

Since:

1.4

See Also:

`Object.hashCode()`, `Collection`, `Set`, `HashSet`, `TreeSet`, `Hashtable`, Serialized Form

Constructor Summary

Constructors

Constructor and Description
`LinkedHashSet()` Constructs a new, empty linked hash set with the default initial capacity (16) and load factor (0.75).
`LinkedHashSet(Collection<? extends E> c)` Constructs a new linked hash set with the same elements as the specified collection.
`LinkedHashSet(int initialCapacity)` Constructs a new, empty linked hash set with the specified initial capacity and the default load factor (0.75).
`LinkedHashSet(int initialCapacity, float loadFactor)` Constructs a new, empty linked hash set with the specified initial capacity and load factor.

Method Summary

Methods inherited from class java.util.HashSet

`add`, `clear`, `clone`, `contains`, `isEmpty`, `iterator`, `remove`, `size`

Methods inherited from class java.util.AbstractSet

`equals`, `hashCode`, `removeAll`

Methods inherited from class java.util.AbstractCollection

`addAll`, `containsAll`, `retainAll`, `toArray`, `toArray`, `toString`

Methods inherited from class java.lang.Object

`finalize`, `getClass`, `notify`, `notifyAll`, `wait`, `wait`, `wait`

Methods inherited from interface java.util.Set

`add`, `addAll`, `clear`, `contains`, `containsAll`, `equals`, `hashCode`, `isEmpty`, `iterator`, `remove`, `removeAll`, `retainAll`, `size`, `toArray`, `toArray`

Constructor Detail

LinkedHashSet

```
public LinkedHashSet(int initialCapacity,
                     float loadFactor)
```

Constructs a new, empty linked hash set with the specified initial capacity and load factor.

Parameters:

`initialCapacity` - the initial capacity of the linked hash set

`loadFactor` - the load factor of the linked hash set

Throws:

`IllegalArgumentException` - if the initial capacity is less than zero, or if the load factor is nonpositive

LinkedHashSet

```
public LinkedHashSet(int initialCapacity)
```

Constructs a new, empty linked hash set with the specified initial capacity and the default load factor (0.75).

Parameters:

`initialCapacity` - the initial capacity of the LinkedHashSet

Throws:

`IllegalArgumentException` - if the initial capacity is less than zero

LinkedHashSet

```
public LinkedHashSet()
```

Constructs a new, empty linked hash set with the default initial capacity (16) and load factor (0.75).

LinkedHashSet

```
public LinkedHashSet(Collection<? extends E> c)
```

Constructs a new linked hash set with the same elements as the specified collection. The linked hash set is created with an initial capacity sufficient to hold the elements in the specified collection and the default load factor (0.75).

Parameters:

`c` - the collection whose elements are to be placed into this set

Throws:

`NullPointerException` - if the specified collection is null

Submit a bug or feature

For further API reference and developer documentation, see Java SE Documentation. That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.