**LinkedHashSet Class in Java with Example**

By Chaitanya Singh | Filed Under: [Java Collections](https://beginnersbook.com/category/java-collections/)

Earlier we have shared tutorials on [HashSet](https://beginnersbook.com/2013/12/hashset-class-in-java-with-example/) and [TreeSet](https://beginnersbook.com/2013/12/treeset-class-in-java-with-example/). [LinkedHashSet](https://docs.oracle.com/javase/6/docs/api/java/util/LinkedHashSet.html) is also an implementation of Set interface, it is similar to the HashSet and TreeSet except the below mentioned differences:

1. HashSet doesn’t maintain any kind of order of its elements.
2. TreeSet sorts the elements in ascending order.
3. LinkedHashSet maintains the insertion order. Elements gets sorted in the same sequence in which they have been added to the Set.

**Example of LinkedHashSet:**

import java.util.LinkedHashSet;

public class LinkedHashSetExample {

public static void main(String args[]) {

// LinkedHashSet of String Type

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

// Adding elements to the LinkedHashSet

lhset.add("Z");

lhset.add("PQ");

lhset.add("N");

lhset.add("O");

lhset.add("KK");

lhset.add("FGH");

System.out.println(lhset);

// LinkedHashSet of Integer Type

LinkedHashSet<Integer> lhset2 = new LinkedHashSet<Integer>();

// Adding elements

lhset2.add(99);

lhset2.add(7);

lhset2.add(0);

lhset2.add(67);

lhset2.add(89);

lhset2.add(66);

System.out.println(lhset2);

}

}

**Output:**

[Z, PQ, N, O, KK, FGH]

[99, 7, 0, 67, 89, 66]

Observe the output: Both types of LinkedHashSet have preserved the insertion order.

### Java LinkedHashSet Class Methods with Examples

Posted by [Ramesh Fadatare](https://www.blogger.com/profile/14691512106162803120) on [November 23, 2018](https://www.javaguides.net/2018/11/java-linkedhshset-class-methods-with-examples.html)

In this article, we will learn the LinkedHashSet class methods with examples.

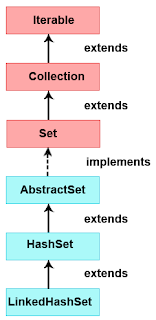
*LinkedHashSet* Class is a Hash table and linked list implementation of the *Set* interface, with predictable iteration order. This implementation differs from HashSet in that it maintains a doubly-linked list running through all of its entries. This linked list defines the iteration ordering, which is the order in which elements were inserted into the set (insertion-order).

This guide covers all the important *LinkedHashSet* class APIs with examples.

## What Will We Learn?

1. Overview of *LinkedHashSet* class
2. Create *LinkedHashSet* Example
3. *LinkedHashSet* Insertion Order Example
4. *LinkedHashSet* remove API's with Example
5. Iterating over a *LinkedHashSet*
6. LinkedHashSet with User-defined objects
7. How to make *LinkedHashSet* thread safe?

## 1. Overview of LinkedHashSet class

[](https://4.bp.blogspot.com/-gCOSruuU-Z8/W1nuuyCV0rI/AAAAAAAAC1M/7t8heIzROQwlYM4LkAoTGLoEbIbN-fdMgCLcBGAs/s1600/linkedhashset-class.png)

* Java *LinkedHashSet* class is a Hash table and Linked list implementation of the *Set* interface.
* Contains unique elements only like HashSet.
* Provides all optional set operations, and permits null elements.
* Maintains insertion order.
* LinkedHashSet 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. Example:

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

## 2. Create a *LinkedHashSet* Example

// Creating a HashSet

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

// Adding new elements to the HashSet

daysOfWeek.add("Monday");

daysOfWeek.add("Tuesday");

daysOfWeek.add("Wednesday");

daysOfWeek.add("Thursday");

daysOfWeek.add("Friday");

daysOfWeek.add("Saturday");

daysOfWeek.add("Sunday");

// Adding duplicate elements will be ignored

daysOfWeek.add("Monday");

System.out.println(daysOfWeek);

Output:

[Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday]

Note that it maintains insertion order.

## *3.* LinkedHashSet Insertion Order Example

package com.ramesh.corejava.devguide.classestypes.set;

import java.util.Iterator;

import java.util.LinkedHashSet;

import java.util.Set;

public class SetInterfaceLinkedHashSetImpl {

public static void main(String[] args) {

linkedHashSetDemo();

}

private static void linkedHashSetDemo() {

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

set.add("element 1");

set.add("element 2");

set.add("element 3");

set.add("element 4");

Iterator<String> iterator = set.iterator();

while (iterator.hasNext()) {

String str = iterator.next();

System.out.println("LinkedHashSet does maintain "

+ " insertion order ---" + str);

}

// loop using java 8

set.forEach(str -> System.out.println("LinkedHashSet does "

+ " maintain insertion order ---" + str));

}

}

Output :

LinkedHashSet does maintain insertion order ---element 1

LinkedHashSet does maintain insertion order ---element 2

LinkedHashSet does maintain insertion order ---element 3

LinkedHashSet does maintain insertion order ---element 4

LinkedHashSet does maintain insertion order ---element 1

LinkedHashSet does maintain insertion order ---element 2

LinkedHashSet does maintain insertion order ---element 3

LinkedHashSet does maintain insertion order ---element 4

## *4.* LinkedHashSet remove API's with Example

### *remove(Object o)*

Remove an element from a *LinkedHashSet* (The *remove()* method returns false if the element does not exist in the *LinkedHashSet*)

Set<Integer> numbers = new LinkedHashSet<>();

numbers.add(2);

numbers.add(3);

numbers.add(4);

numbers.add(5);

numbers.add(6);

numbers.add(7);

numbers.add(8);

numbers.add(9);

numbers.add(10);

System.out.println("numbers : " + numbers);

boolean isRemoved = numbers.remove(10);

System.out.println("After remove(10) => " + numbers);

### *removeAll(Collection<?> c)*

Remove all elements belonging to a given collection from a *LinkedHashSet*.

Set<Integer> numbers = new LinkedHashSet<>();

numbers.add(2);

numbers.add(3);

numbers.add(4);

numbers.add(5);

numbers.add(6);

numbers.add(7);

numbers.add(8);

numbers.add(9);

numbers.add(10);

List<Integer> perfectSquares = new ArrayList<>();

perfectSquares.add(4);

perfectSquares.add(9);

numbers.removeAll(perfectSquares);

System.out.println("After removeAll(perfectSquares) => " + numbers);

### *removeIf(Predicate<? super Integer> filter)*

Remove all elements matching a given predicate

numbers.removeIf(num -> num % 2 == 0);

System.out.println("After removeIf() => " + numbers);

### *clear()*

Remove all elements from *LinkedHashSet*(clear it completely)

numbers.clear();

System.out.println("After clear() => " + numbers);

## 5. Iterating over a *LinkedHashSet*

The following example shows different ways of iterating over a HashSet

* Iterate over a *LinkedHashSet* using Java 8 forEach and lambda expression.
* Iterate over a *LinkedHashSet* using *iterator().*
* Iterate over a *LinkedHashSet* using iterator() and Java 8 *forEachRemaining()* method.
* Iterate over a *LinkedHashSet* using a simple for-each loop.

### *iterator()*

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

list.add("element 1");

list.add("element 2");

list.add("element 3");

list.add("element 4");

// using Iterator

Iterator<String> iterator = list.iterator();

while (iterator.hasNext()) {

String str = iterator.next();

System.out.println(" only forward direction ---" + str);

}

### Advance *for()* loop

// Using advanced for loop

for (String str : list) {

System.out.println(" only forward direction ---" + str);

}

### *forEachRemaining()*

// Java 8

list.forEachRemaining(str -> System.out.println(" only forward direction ---" + str));

### *forEach()*

// Java 8

list.forEach(str -> System.out.println(" only forward direction ---" + str));

## *6.* LinkedHashSet with User-defined objects

This example shows how to create a *LinkedHashSet* of user-defined objects and it maintains insertion order.

import java.util.HashSet;

import java.util.Objects;

import java.util.Set;

class Customer {

private long id;

private String name;

public Customer(long id, String name) {

this.id = id;

this.name = name;

}

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

// Two customers are equal if their IDs are equal

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Customer customer = (Customer) o;

return id == customer.id;

}

@Override

public int hashCode() {

return Objects.hash(id);

}

@Override

public String toString() {

return "Customer{" +

"id=" + id +

", name='" + name + '\'' +

'}';

}

}

public class HashSetUserDefinedObjectExample {

public static void main(String[] args) {

Set<Customer> customers = new LinkedHashSet<>();

customers.add(new Customer(101, "Rajeev"));

customers.add(new Customer(102, "Sachin"));

customers.add(new Customer(103, "Chris"));

/\*

HashSet will use the `equals()` & `hashCode()` implementations

of the Customer class to check for duplicates and ignore them

\*/

customers.add(new Customer(101, "Rajeev"));

System.out.println(customers);

}

}

## 7. How to make LinkedHashSet thread safe?

This class implementation is not synchronized so it is not thread-safe. If multiple threads access a hash set concurrently, and at least one of the threads modifies the set, it must be synchronized externally. *LinkedHashSet* must be synchronized externally. Example :

// HashSet is not synchronized

private static void synchronizedHashSetDemo() {

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

Set<String> synchronizedSet = Collections.synchronizedSet(set);

}