CSCI 2120:
Software Design & Development II

UNIT 2: Collections Framework & Generics

LinkedHashSet

Overview

Introduction

LinkedHashSet in Java is a concrete class that implements Set interface and extends HashSet class with a doubly linked list implementation. It internally uses a linked list to store the elements in the set.

LinkedHashSet class is the same as HashSet class, except that it maintains the ordering of elements in the set in which they are inserted. LinkedHashSet not only uses a hash table for storing elements but also maintains a double-linked list of the elements in the order during iteration.

In simple words, elements in the HashSet are not ordered, but elements in the LinkedHashSet can be retrieved in the same order in which they were inserted into the set.

Hierarchy of LinkedHashSet class in Java

The hierarchy diagram of Java LinkedHashSet is shown in the below figure.

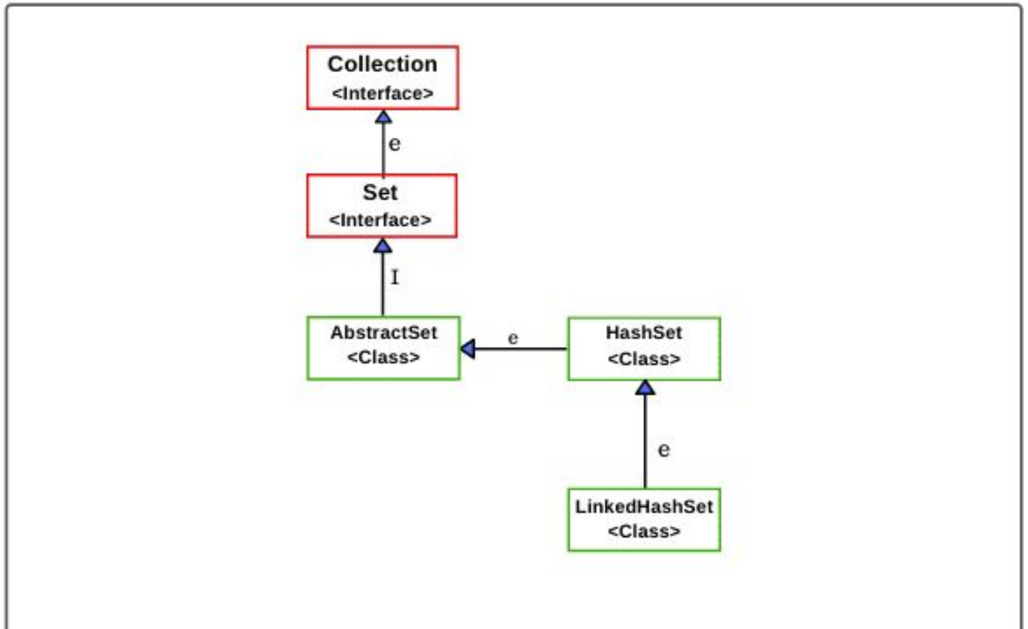


Fig: Java LinkedHashSet Hierarchy Diagram

LinkedHashSet class does not define any extra methods of its own. Since LinkedHashSet class extends HashSet and implements the Set interface,

Therefore, all the methods defined by the Set interface and HashSet class can be used while using LinkedHashSet class.

LinkedHashSet Class Declaration

LinkedHashSet is a generic class that has declaration given below:

```
class LinkedHashSet<T> extends HashSet<T>
```

Here, T defines the generic type parameter that represents the data type of elements that the LinkedHashSet will hold

Features of LinkedHashSet

The important features of LinkedHashSet class:

1. LinkedHashSet class contains unique elements like HashSet. It does not allow to insert of duplicate elements. If we try to add a duplicate element, it will fail and the iteration order of the set is not modified.

2. LinkedHashSet class permits to insert null element.

3. LinkedHashSet class in Java is non-synchronized. That means it is not thread-safe.

4. LinkedHashSet class preserve the insertion order of elements

5. It is slightly slower than HashSet.

6. Linked hash set is very efficient for insertion and deletion of elements.

Constructors of LinkedHashSet

1. LinkedHashSet( ): This constructor is used to create an empty LinkedHashSet. It is a default LinkedHashSet. The general syntax to create LinkedHashSet is as follows:

```
LinkedHashSet<T> lhset = new LinkedHashSet<T>();
```

Constructors of LinkedHashSet

2. LinkedHashSet(Collection c): This constructor is used to initialize the LinkedHashSet with elements of collection c. The general syntax is as follow:

```
LinkedHashSet<T> lhset = new LinkedHashSet<T>(Collection c);
```

Constructors of LinkedHashSet

3. LinkedHashSet(int initialCapacity): This constructor is used to create LinkedHashSet with initializing the capacity of the linked hash set. It takes an integer value to initialize capacity. The general syntax to create linked hash set in Java is given below:

```
LinkedHashSet<T> lhset = new LinkedHashSet<T>(int size);
```

Constructors of LinkedHashSet

4. LinkedHashSet(int initialCapacity, float loadFactor): This constructor is used to create LinkedHashSet with initializing both the capacity and load factor of the linked hash set.

```
LinkedHashSet<T> lhset = new LinkedHashSet<T>(int initialCapacity, float loadFactor);
```

LinkedHashSet Examples

In this section, we will take some example program where we will perform a few frequently used operations on the Java LinkedHashSet.

Example 1: Adding Elements

1. **Adding elements:** Let’s create a program where we will perform operations such as adding, checking the size of LinkedHashSet, etc.

Example 1: Adding Elements

```java
import java.util.LinkedHashSet;
public class LinkedHashSetTester1 {
   public static void main(String[] args) {
       // Create a Linked hash set of generic type.
       LinkedHashSet<String> lhset= new LinkedHashSet<String>();

       // Checking the size of LinkedHashSet before adding elements.
       int size = lhset.size();
       System.out.println("Size of LinkedHashSet before adding elements: " +size);

       // Adding elements in the linked hash set.
       lhset.add("Red"); // lhset.size() is 1.
       lhset.add("Green"); // lhset.size() is 2.
       lhset.add("Yellow"); // lhset.size() is 3.
       lhset.add("Blue"); // lhset.size() is 4.
       lhset.add("Orange"); // lhset.size() is 5.

       System.out.println("Elements in Set: " +lhset);
       int size2 = lhset.size();
       System.out.println("Size of LinkedHashSet after adding elements: " +size2);

       // Adding duplicate elements that already exist in set.
       lhset.add("Red"); // lhset.size() is still 5.
       lhset.add("Yellow"); // lhset.size() is still 5.

       // Create another set of String type.
       LinkedHashSet<String> lhset2 = new LinkedHashSet<String>();
       lhset2.add("Brown");
       lhset2.add(null);

       // Adding elements of set2 into set.
       lhset.addAll(lhset2);
       System.out.println("Elements in Set after adding: " +lhset);
   }
}
```

Example 1: Adding Elements

Output:

```
Size of LinkedHashSet before adding elements: 0
Elements in Set: [Red, Green, Yellow, Blue, Orange]
Size of LinkedHashSet after adding elements: 5
Elements in Set after adding: [Red, Green, Yellow, Blue, Orange, Brown, null]
```

Example 2: Removing Elements

2. **Removing element:** Let’s create another program where we will remove an element from the linked hash set.

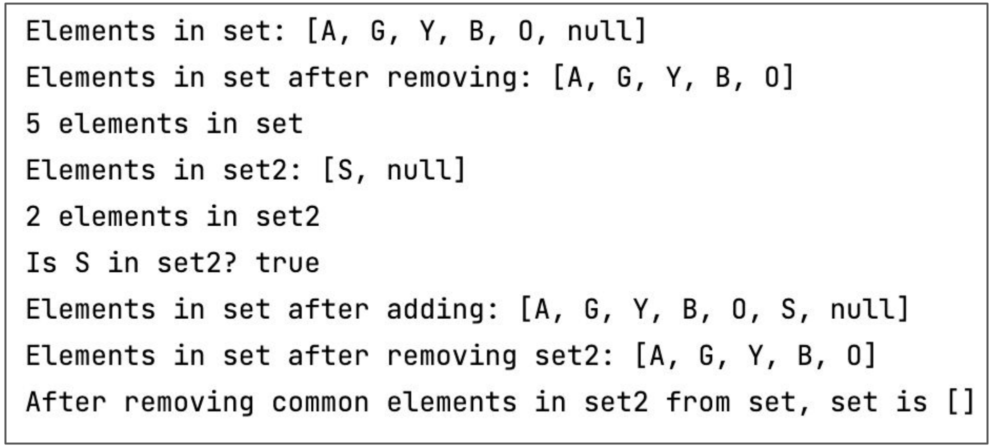
Example 2: Removing Elements

```java
import java.util.LinkedHashSet;
public class LinkedHashSetTester2 {
   public static void main(String[] args) {
       // Create a Linked hash set of generic type.
       LinkedHashSet<String> set= new LinkedHashSet<String>();
       // Adding elements in the linked hash set.
       set.add("A");
       set.add("G");
       set.add("Y");
       set.add("B");
       set.add("O");
       set.add(null);
       System.out.println("Elements in set: " +set);
       // Remove a string element from linked hash set.
       set.remove(null);
       System.out.println("Elements in set after removing: " +set);
       System.out.println(set.size()+ " elements in set");
       // Create another linked hash set of String type.
       LinkedHashSet<String> set2 = new LinkedHashSet<String>();
       set2.add("S");
       set2.add(null);
       System.out.println("Elements in set2: " +set2);
       System.out.println(set2.size()+ " elements in set2");
       System.out.println("Is S in set2? " +set2.contains("S"));

       set.addAll(set2);
       System.out.println("Elements in set after adding: " +set);
       set.removeAll(set2);
       System.out.println("Elements in set after removing set2: " +set);
       set.retainAll(set2);
       System.out.println("After removing common elements in set2 " + "from set, set is " + set);
   }
}
```

Example 2: Removing Elements

Output:



```
Elements in set: [A, G, Y, B, O, null]
Elements in set after removing: [A, G, Y, B, O]
5 elements in set
Elements in set2: [S, null]
2 elements in set2
Is S in set2? true
Elements in set after adding: [A, G, Y, B, O, S, null]
Elements in set after removing set2: [A, G, Y, B, O]
After removing common elements in set2 from set, set is []
```

Example 3: Removing Duplicate Elements

3. **Removing duplicate elements:** Let’s make a program where we will remove duplicate numbers from ArrayList using LinkedHashSet.

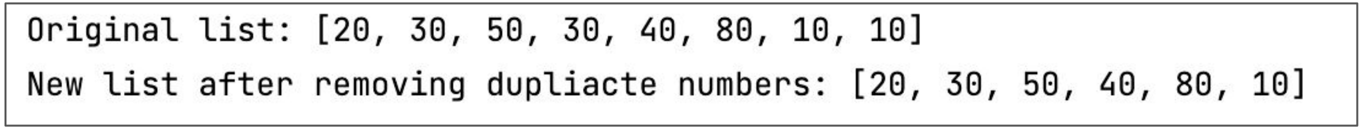
Example 3: Removing Duplicate Elements

```java
import java.util.ArrayList;
import java.util.LinkedHashSet;
public class LinkedHashSetTester3 {
   public static void main(String[] args) {
       int[] num = {20, 30, 50, 30, 40, 80, 10, 10};
       ArrayList<Integer> ar = new ArrayList<Integer>();

       // Adding numbers to the array list.
       for(int i = 0; i < num.length; i++) {
           ar.add(num[i]);
       }
       System.out.println("Original list: " +ar);
       LinkedHashSet<Integer> lhset = new LinkedHashSet<>(ar);
       System.out.println("New list after removing duplicate numbers: " +lhset);
   }
}
```

Example 3: Removing Duplicate Elements

Output:



```
Original list: [20, 30, 50, 30, 40, 80, 10, 10]
New list after removing dupliacte numbers: [20, 30, 50, 40, 80, 10]
```

Key point:

As you can observe in the output, after removing duplicate elements from ArrayList, LinkedHashSet maintains the insertion order of elements in which they had been inserted into the list.

Example 4: Iterating LinkedHashSet

4. **Iterating LinkedHashSet:** Let’s take an example program where we will iterate elements of LinkedHashSet using iterator() method and enhanced for loop.

Example 4: Iterating LinkedHashSet

```
import java.util.Iterator;
import java.util.LinkedHashSet;
public class LinkedHashSetTester4 {
   public static void main(String[] args) {
       LinkedHashSet<String> lhset = new LinkedHashSet<>();
       lhset.add("New York");
       lhset.add("Dhanbad");
       lhset.add("Sydney");
       lhset.add("Cape Town");
       lhset.add("London");

       // Iterating elements of LinkedHashSet using iterator() method.
       System.out.println("Iteration using iterator");
       Iterator<String> itr = lhset.iterator();
       while(itr.hasNext()) {
           System.out.println(itr.next());
       }
       System.out.println();

       // Iterating elements of LinkedHashSet using enhanced for loop
       System.out.println("Iteration using enhanced for loop");
       for (String s : lhset)
           System.out.print(s + " ");
       System.out.println();
   }
}
```

Example 4: Iterating LinkedHashSet

Output:

```
Iteration using iterator
New York
Dhanbad
Sydney
Cape Town
London

Iteration using enhanced for loop
New York Dhanbad Sydney Cape Town London
```

Example 5: Adding custom objects

5. **Adding custom objects:** Let’s take an example program where we will add custom objects of type Student into LinkedHashSet and iterate it.

Example 5: Adding custom objects

```java
import java.util.LinkedHashSet;
public class LinkedHashSetTester5 {
   public static void main(String[] args)
   {
       LinkedHashSet<Person> lhset = new LinkedHashSet<>();

       // Creating objects of Students.
       Person st1 = new Person("John", 2345, "New York");
       Person st2 = new Person("Deep", 1234, "Dhanbad");
       Person st3 = new Person("Ricky", 7583, "Cape Town");

       // Adding elements (object references) into LinkedHashSet.
       lhset.add(st1);
       lhset.add(st2);
       lhset.add(st3);

       // Traversing linked hash set.
       for(Person s:lhset){
           System.out.println("Name: " +s.name+" "+ "Id: " +s.id+" "+"City: "+s.city);
       }
   }
}

class Person {
   String name;
   int id;
   String city;

   Person(String name, int id, String city){
       this.name = name;
       this.id = id;
       this.city = city;
   }
}
```

Example 5: Adding custom objects

Output:

```
Name: John Id: 2345 City: New York
Name: Deep Id: 1234 City: Dhanbad
Name: Ricky Id: 7583 City: Cape Town
```

When to use LinkedHashSet in Java?

LinkedHashSet can be used when you do not want duplicate elements (i.e. want to remove duplicate elements) and want to maintain order in which elements are inserted.

If you want to impose different orders such as increasing or decreasing order, you can use TreeSet class that you will learn in the next lecture.

Which is better to use: HashSet or LinkedHashSet?

If you do not require to maintain order in which elements are inserted then use HashSet which is faster and more efficient than LinkedHashSet.

END