CSCI 2120:
Software Design & Development II

UNIT 2: Collections Framework & Generics

HashSet

Overview

1. Introduction
2. Hierarchy of HashSet in Java
3. HashSet class declaration
4. Features of HashSet
5. Constructors of HashSet class
6. HashSet Methods
7. Examples
8. When to use HashSet in Java

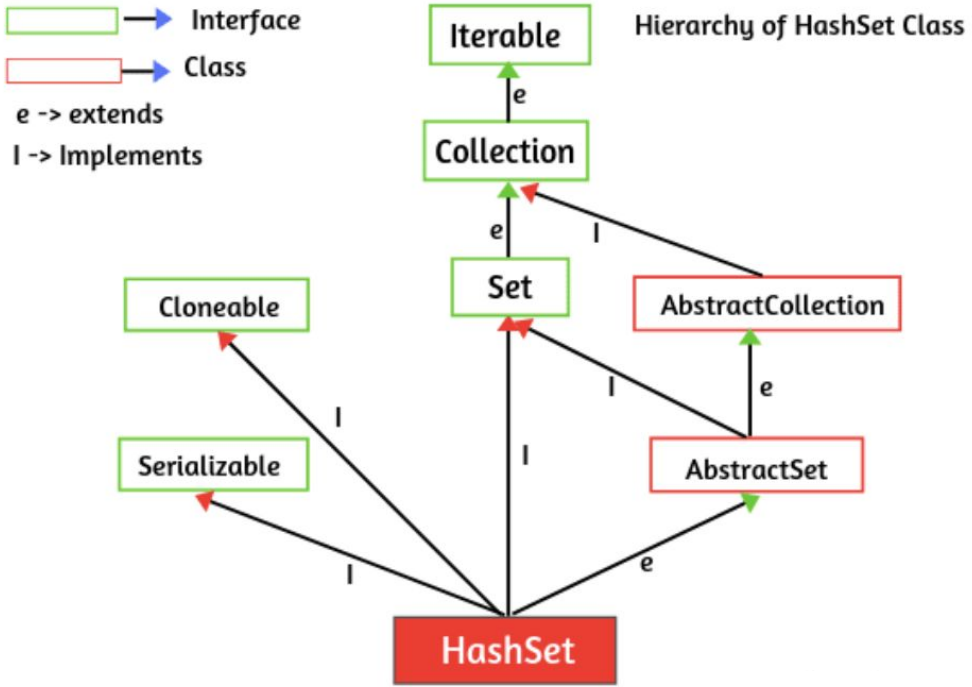
Introduction

HashSet in Java is an unordered collection of elements (objects) that contains only unique elements.

It internally uses Hashtable data structure to store a set of unique elements. It is much faster and gives constant-time performance for searching and retrieving elements.

HashSet was introduced in Java 1.2 version and it is present in java.util.HashSet package.

Hierarchy of HashSet in Java



HashSet class extends AbstractSet class and implements the Set interface. The AbstractSet class itself extends AbstractCollection class.

It also implements Cloneable and Serializable marker interfaces. The hierarchy diagram of Java HashSet is shown in the below figure.

HashSet class declaration

HashSet is a concrete generic class that can be declared in general form as below:

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

Here, E defines the type of Elements that the Set will hold.

Features of HashSet

There are several important features of HashSet:

1. The underlying data structure of HashSet is Hashtable. A hash table stores data by using a mechanism called **hashing**.

2. HashSet does not allow duplicate elements. If we try to add a duplicate element in HashSet, the older element would be overwritten.

3. It allows only one null element. If we try to add more than one null element, it would still return only one null value.

4. HashSet does not maintain any order in which elements are added. The order of the elements will be unpredictable. It will return in any random order.

5. It is much faster due to the use of hashing technique and gives constant-time performance for adding (insertion), retrieval, removal, contains, and size operations. Hashing provides constant execution time for methods like add(), remove(), contains(), and size() even for the large set.

6. HashSet class is not synchronized which means it is not thread-safe. Therefore, multiple threads can use the same HashSet object at the same time and will not give the deterministic final output.

If you want to synchronize HashSet, use Collections.`synchronizeSet()` method.

7. The iterator returned by HashSet class is fail-fast which means any modification that happened in the HashSet during iteration, will throw ConcurrentModificationException.

Constructors of HashSet class

1. HashSet(): It constructs an empty HashSet (i.e, default HashSet). The default capacity is 16. The syntax to create Hashset object is as follows:

```
HashSet hset = new HashSet();

// Generic form. T is the type of object that HashSet will hold.
HashSet<T> hset = new HashSet<T>();
```

Constructors of HashSet class

2. HashSet(int initialCapacity): It initializes the capacity of HashSet. When the set capacity reaches full and a new element is added, the capacity of the hash set is expanded automatically.

The internal structure will double in size before adding a new element. The general syntax in generic form is as follows:

```
HashSet<T> hset = new HashSet<T>(int initialCapacity);
```

Constructors of HashSet class

3. HashSet(int initialCapacity, float fillRatio): This form of constructor initializes capacity and fill ratio (also called load factor or load capacity) of the hash set.

When the number of elements is greater than capacity of HashSet, the size of the HashSet is grown automatically by multiplying capacity with load factor.

The default value of the load factor is 0.75. The value of the fill ratio ranges from 0.0 to 1.0.
The general syntax to create an object of HashSet with initial capacity and load factor is as follows:

```
HashSet<T> hset = new HashSet<T>(int initialCapacity, float loadFactor);

//For example:
HashSet<Integer> hset = new HashMap<Integer>(30, 0.7f);
```

Constructors of HashSet class

4. HashSet(Collection c): It initializes HashSet by using elements of c. This constructor acts as a copy constructor. It copies elements from one collection into the newly created collection. We cannot provide a custom initial capacity or fill ratio.

If the original collection had duplicate elements, only one duplicate element will be allowed in the final created set.

Note: Constructors that do not take a load factor, 0.75 is used.

HashSet Methods

The java HashSet class does not define any additional methods. It inherits methods of its parent classes and methods of the implemented interface.

The various methods provided by its superclasses and interfaces are as follows:

HashSet Methods

Method	Description
boolean add(Object o)	This method is used to add the specified element in this set. It returns true if set does not already contain a specified element.
boolean addAll(Collection c)	This method is used to add a group of elements from another collection to the set. Each element in the collection will be added to the current set via calling add() method on each element. It returns true if the current set changes. If no elements are added, false is returned. A UnsupportedOperationException will be thrown when a current set does not support adding elements.

HashSet Methods

Method	Description
boolean remove(Object o)	This method is used to remove the specified element from the set. To remove an element, set internally use equals() method for each element. If the element is found, element is removed from set and returns true. If not found, false is returned. If the removal is not supported, you will get UnsupportedOperationException.
void clear	This method is used to remove all elements from a set. It returns nothing.

HashSet Methods

Method	Description
boolean contains(Object element)	To check a specific element is present or not in the set, use contains() method. If the specified element is found in the set, it returns true otherwise false. The equality checking is done through the element's equal method.
int size()	If you wish to know how many elements are in a set, call size() method.
boolean isEmpty()	If you wish to check whether HashSet contains elements or not. Call isEmpty() method.

HashSet Methods

Method	Description
Object clone()	Since HashSet implements cloneable interface. So, we can create a shallow copy of that hash set by calling clone() method of HashSet.
boolean equals(Object o)	The HashSet class defines equality by using equals() method on each element within the set.
int hashCode()	The HashSet class overrides the hashCode() method to define an appropriate hash code value for the set. It returns the same hash code by sum up all hash codes of all the elements.

Example 1: Add elements to HashSet

1. Let’s take an example program where we add elements to Set. We will also add duplicate elements but HashSet will not store them again because hashset does not allow duplicate data.

In this example program, we will add null element into the set. Look at the program source code below.

Example 1: Add elements to HashSet

```
import java.util.HashSet;
public class HashSetTester1 {
   public static void main(String[] args) {
       // Create a HashSet object.
       HashSet<String> set = new HashSet<String>(); // An empty hash set.

       // Adding elements to HashSet.
       set.add("First");
       set.add("Second");
       set.add("Third");
       set.add("Fourth");
       set.add("Fifth");

       // Adding duplicate elements that will be ignored.
       set.add("First");
       set.add("Third");

       // Adding of null elements.
       set.add(null);
       set.add(null); // Ignored.
       System.out.println("Unordered and No Duplicate HashSet Elements");
       System.out.println(set);
   }
}
```

Example 1: Add elements to HashSet

Output:

```
Unordered and No Duplicate HashSet Elements
[null, Second, Third, First, Fourth, Fifth]
```

Key point:

As you can observe in the output that the set does not maintain the same order of elements in which they have been inserted.

Example 2: Add existing collection to HashSet

2. Let’s take an example program where we will see how to add elements from the existing collection to HashSet in Java.

Example 2: Add existing collection to HashSet

```
import java.util.ArrayList;
import java.util.HashSet;
public class HashSetTester2 {
   public static void main(String[] args) {
       // Create an ArrayList object.
       ArrayList<String> al = new ArrayList<String>();
       al.add("Monday");
       al.add("Tuesday");
       al.add("Wednesday");
       al.add("Thursday");
       al.add("Friday");

       // Adding duplicate elements.
       al.add("Monday");
       al.add("Friday");
       System.out.println("Original Elements Order ");
       System.out.println(al);

       // Create HashSet object.
       HashSet<String> hset = new HashSet<String>();

       // Call addAll() method for adding all elements from existing collection to HashSet.
       hset.addAll(al);
       System.out.println("Unordered HashSet Elements without Duplicate elements");
       System.out.println(hset);
   }
}
```

Example 2: Add existing collection to HashSet

Output:

```
Original Elements Order
[Monday, Tuesday, Wednesday, Thursday, Friday, Monday, Friday]
Unordered HashSet Elements without Duplicate elements
[Monday, Thursday, Friday, Wednesday, Tuesday]
```

Example 3: Remove elements from HashSet

3. Let’s take an example program where we will see how to remove a specific element from a HashSet and remove all elements available in a set.

Example 3: Remove elements from HashSet

```
import java.util.HashSet;
public class HashSetTester3 {
   public static void main(String[] args) {
       HashSet<Integer> hset = new HashSet<Integer>();
       hset.add(5);
       hset.add(10);
       hset.add(15);
       hset.add(20);
       System.out.println("Initial list of elements");
       System.out.println(hset);

       // Removing a specific element from HashSet.
       hset.remove(10);
       System.out.println("List of elements after removing 10");
       System.out.println(hset);

       HashSet<Integer> hset2 = new HashSet<Integer>();
       hset2.add(10);
       hset2.add(25);
       hset.addAll(hset2);
       System.out.println("List of Elements after adding elements from existing collection");
       System.out.println(hset);

       // Removing all new elements from HashSet.
       hset.removeAll(hset2);
       System.out.println("List of Elements after removing elements from hset2");
       System.out.println(hset);

       // Removing all elements available in HashSet.
       hset.clear();
       System.out.println("After invoking clear() method: "+hset);
   }
}
```

Example 3: Remove elements from HashSet

Output:

```
Initial list of elements
[20, 5, 10, 15]
List of elements after removing 10
[20, 5, 15]
List of Elements after adding elements from existing collection
[20, 5, 25, 10, 15]
List of Elements after removing elements from hset2
[20, 5, 15]
After invoking clear() method: []
```

Example 4: Size of HashSet

4. Let’s create a program where we will check the number of elements in Java HashSet. We will also verify that HashSet is empty or not.

Example 4: Size of HashSet

```
import java.util.HashSet;
import java.util.Set;
public class HashSetTester4 {
   public static void main(String[] args) {
       Set<String> pCountry = new HashSet<String>();

       // Check that HashSet is empty or not.
       System.out.println("Is popularCountries set empty? : " + pCountry.isEmpty());
       System.out.println("Number of countries in HashSet before adding: " +pCountry.size());
       pCountry.add("INDIA");
       pCountry.add("USA");
       pCountry.add("UK");
       pCountry.add("FRANCE");

       // Find size of HashSet.
       System.out.println("Number of countries in HashSet after adding: " + pCountry.size());
   }
}
```

Example 4: Size of HashSet

Output:

```
Is popularCountries set empty? : true
Number of countries in HashSet before adding: 0
Number of countries in HashSet after adding: 4
```

Example 5: Check whether HashSet contains an element

5. Let’s create a program where we check the existing element in HashSet. We will use contains() method for this.

Example 5: Check whether HashSet contains an element

```
import java.util.HashSet;
import java.util.Set;
public class HashSetTester5 {
   public static void main(String[] args) {
       Set<String> set = new HashSet<String>();
       System.out.println("Is set empty? : " + set.isEmpty());
       System.out.println("Number of elements in HashSet before adding: " +set.size());

       set.add("Dollar");
       set.add("Indian Rupee");
       set.add("Euro");
       set.add("Yen");

       System.out.println("List of Elements in set");
       System.out.println(set);
       System.out.println("Number of elements in the HashSet after adding: " + set.size());

       // Call contains() method to check an element exists in set or not.
       if(set.contains("Dollar")){
           System.out.println("Does Element 'Dollar' exist in set?");
           System.out.println("Yes, Element 'Dollar' exists in set");
       }
       else{
           System.out.println("No, Element 'Dollar' does not exist in set");
       }
       System.out.println("Does Element 'Dinar' exist in set?");
       if(set.contains("Dinar")){
           System.out.println("Yes, Element 'Dinar' exists in set ");
       }
       else {
           System.out.println("No, Element 'Dinar' does not exist in set");
       }
   }
}
```

Example 5: Check whether HashSet contains an element

Output:

```
Is set empty? : true
Number of elements in HashSet before adding: 0
List of Elements in set
[Yen, Dollar, Indian Rupee, Euro]
Number of elements in the HashSet after adding: 4
Does Element 'Dollar' exist in set?
Yes, Element 'Dollar' exists in set
Does Element 'Dinar' exist in set?
No, Element 'Dinar' does not exist in set
```

Example 6: Create User-defined Object of HashSet

6. Let’s create a program where we create a user-defined object of HashSet in Java.

Example 6: Create User-defined Object of HashSet

```
import java.util.HashSet;
public class HashSetTester6 {
   public static void main(String[] args) {
       // Create a user-defined HashSet object of type Student.
       HashSet<Student> hset = new HashSet<Student>();
       // Create objects of Student class and pass the parameters to their constructors.
       Student s1 = new Student("John", "RSVM", 0012);
       Student s2 = new Student("Shubh", "DPS", 1234);
       Student s3 = new Student("Ricky", "DAV", 9876);
       // Adding elements to HashSet and pass reference variables s1, s2, s3.
       hset.add(s1);
       hset.add(s2);
       hset.add(s3);
       // Traversing HashSet.
       for(Student s:hset) {
           System.out.println(s.name+" "+s.sName+" "+s.id);
       }
   }
}

class Student {
   String name, sName;
   int id;
   public Student(String name, String sName, int id) {
       this.name = name;
       this.sName = sName;
       this.id = id;
   }
}
```

Example 6: Create User-defined Object of HashSet

Output:

```
John RSVM 10
Shubh DPS 1234
Ricky DAV 9876
```

When to use HashSet in Java

HashSet in Java is used when

1. We don’t want to store duplicate elements.
2. We want to remove duplicate elements from the list.
3. HashSet is preferred when add and remove operations are more abundant as opposed to get operations.
4. We are not working in a multithreading environment.

END