

Java HashSet class

📅 Last Updated: December 26, 2020 👤 By: Lokesh Gupta 📁 Java Collections 🔗 Java HashSet

Java HashSet class implements the **Set** interface, backed by a hash table(actually a **HashMap** instance). It does not offer any guarantees as to the iteration order, and allows **null** element.

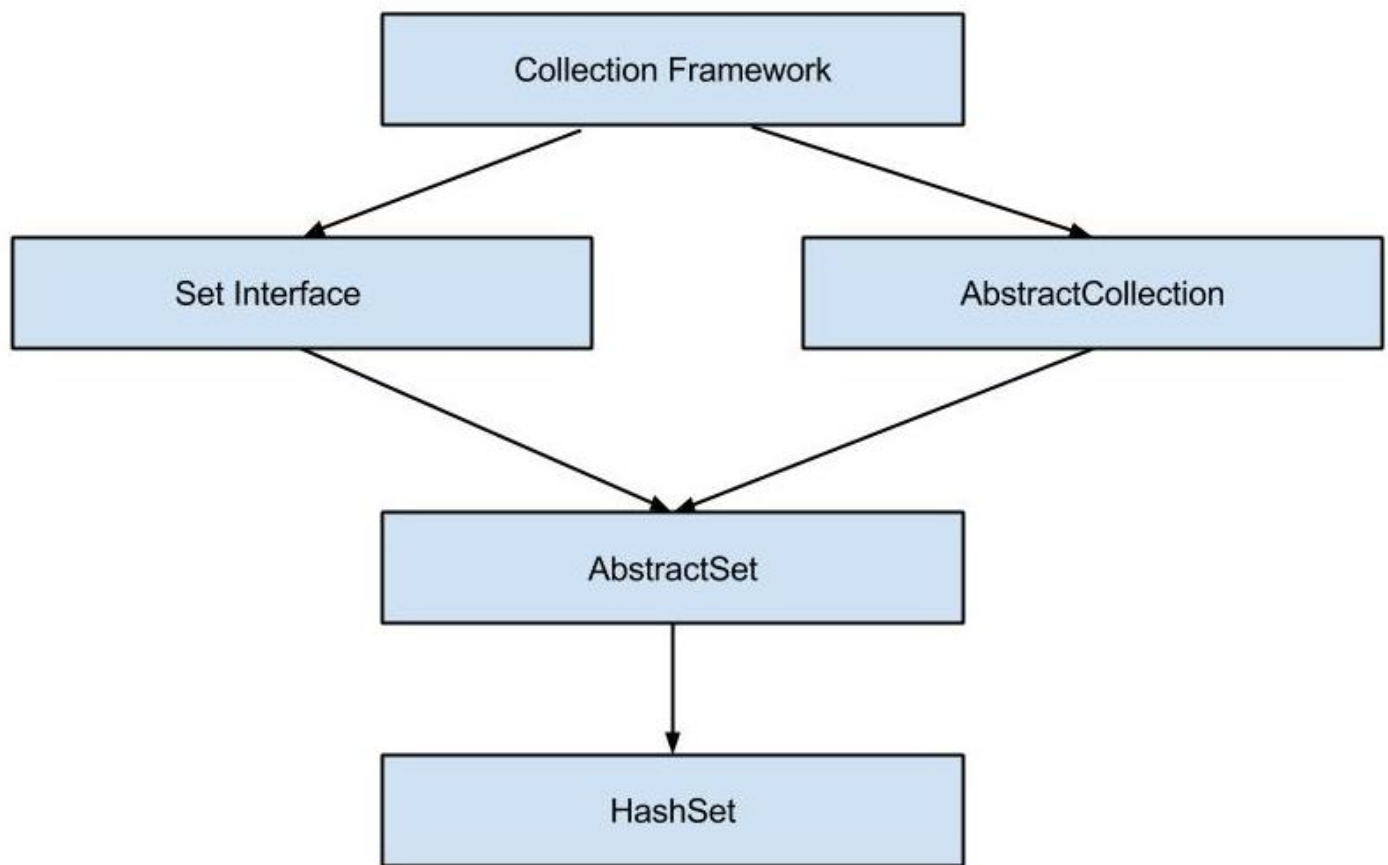
Table of Contents

1. [HashSet Hierarchy](#)
2. [HashSet Features](#)
3. [HashSet Constructors](#)
4. [HashSet Methods](#)
5. [HashSet Example](#)
6. [HashSet Usecases](#)
7. [HashSet Performance](#)
8. [Conclusion](#)

1. HashSet Hierarchy

The **HashSet** class extends **AbstractSet** class which implements **Set** interface. The **Set** interface inherits **Collection** and **Iterable** interfaces in hierarchical order.

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



HashSet Hierarchy

2. HashSet Features

- It implements **Set** Interface.
- Duplicate values are not allowed in HashSet.
- One NULL element is allowed in HashSet.
- It is un-ordered collection and makes no guarantees as to the iteration order of the set.
- This class offers constant time performance for the basic operations(add, remove, contains and size).
- HashSet 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 HashSet())** method to get the synchronized hashset.

- 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.
- HashSet also implements **Serializable** and **Cloneable** interfaces.

2.1. Initial Capacity

The initial capacity means the number of buckets (in backing HashMap) when hashset 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 its constructor **HashSet(int initialCapacity)**.

2.2. Load Factor

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

This is called **threshold** and is equal to $(\text{DEFAULT_LOAD_FACTOR} * \text{DEFAULT_INITIAL_CAPACITY})$. When HashSet elements count exceed this threshold, HashSet is resized and new capacity is double the previous capacity.

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

3. HashSet Constructors

The HashSet has four types of constructors:

1. **HashSet()**: initializes a default HashSet instance with the default initial capacity (16) and default load factor (0.75).

2. **HashSet(int capacity)**: initializes a HashSet with a specified capacity and default load factor (0.75).
3. **HashSet(int capacity, float loadFactor)**: initializes HashSet with specified initial capacity and specified load factor.
4. **HashSet(Collection c)**: initializes a HashSet with same elements as the specified collection.

4. HashSet 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 hashset.
3. **public boolean contains(Object o)** : returns **true** if the hashset contains the specified element, otherwise **false**.
4. **public boolean isEmpty()** : returns **true** if hashset contains no element, otherwise **false**.
5. **public int size()** : returns the number of elements in the hashset.
6. **public Iterator<E> iterator()** : returns an iterator over the elements in this hashset. The elements are returned from iterator in no specific order.
7. **public boolean remove(Object o)** : removes the specified element from the hashset if it is present and return **true**, else returns **false**.
8. **public boolean removeAll(Collection<?> c)** : remove all the elements in the hashset that are part of the specified collection.
9. **public Object clone()** : returns a shallow copy of the hashset.
10. **public Spliterator<E> spliterator()** : creates a late-binding and fail-fast **Spliterator** over the elements in this hashset.

5. Java HashSet Example

5.1. HashSet add, remove, iterator example

Java HashSet Example

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

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

System.out.println(hashSet);

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

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

//5. Iterate over values
Iterator<String> itr = hashSet.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 HashSet to Array Example

Java example to convert a hashset to [array](#) using **toArray()** method.

Java HashSet Example

```
HashSet<String> hashSet = new HashSet<>();

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

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

hashSet.toArray(values);

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

Program Output.

Console

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

5.3. Convert HashSet to ArrayList Example

Java example to convert a hashset to [arraylist](#) using [Java 8 stream API](#).

Java HashSet Example

```
HashSet<String> hashSet = new HashSet<>();

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

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

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

Program Output.

Console

[A, B, C, D, E]

6. HashSet Usecases

HashSet is very much like **ArrayList** class. It additionally restrict the duplicate values. So when we have a requirement where we want to store only distinct elements, we can choose HashSet.

A real-life usecase for HashSet can be storing data from stream where the stream may contain duplicate records, and we are only interested in distinct records.

Another usecase can be finding distinct words in a given sentence.

7. Java HashSet Performance

- HashSet 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.
- Iterating over this set requires time proportional to the sum of the HashSet instance's size (the number of elements) plus the "capacity" of the backing HashMap instance (the number of buckets). Thus, it's very important not to set the initial capacity too high (or the load factor too low) if iteration performance is important.

8. Conclusion

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

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

Happy Learning !!

Reference:

[HashSet Java Docs](#)

Was this post helpful?

Let us know if you liked the post. That's the only way we can improve.

Yes

No

Recommended Reading:

1. [How to convert HashSet to ArrayList in Java](#)
2. [\[Solved\]: javax.xml.bind.JAXBException: class java.util.ArrayList nor any of its super class is known to this context](#)
3. [Java LinkedHashSet class](#)
4. [Java CopyOnWriteArraySet class](#)
5. [Java TransferQueue – Java LinkedTransferQueue class](#)
6. [Java TreeMap class](#)
7. [Java Hashtable class](#)
8. [Java TreeSet class](#)
9. [Java LinkedList class](#)
0. [Java CopyOnWriteArrayList class](#)

Join 7000+ Awesome Developers

Get the latest updates from industry, awesome resources, blog updates and much more.

Email Address

Subscribe

** We do not spam !!*

5 thoughts on “Java HashSet class”

Anand Reddy

June 12, 2020 at 10:01 pm

Please have an example of un-ordered collection as mentioned in the post for better understanding. The example given shows it all an ordered collection.

[Reply](#)

sai kishore beeram

April 7, 2020 at 9:49 pm

in HashSet Usecases, you might want to specify again that HashSet doesnt maintain the order of insertion (un ordered).

[Reply](#)

Manish Gour

February 24, 2020 at 9:35 pm

Small improvement in point 3. HashSet Constructors

3.2. HashSet(int capacity): initializes a HashSet with a specified capacity and load factor (0.75).

Better to use word "Default" for better understanding, so it will become

3.2. HashSet(int capacity): initializes a HashSet with a specified capacity and "default" load factor (0.75).

[Reply](#)**Lokesh Gupta**[February 25, 2020 at 11:02 pm](#)


Thanks for the feedback. Updated the post.

[Reply](#)**Nisha**[December 5, 2019 at 12:56 pm](#)

why do we want to convert hashset to array. I mean when do we use in real time

[Reply](#)

Leave a Comment

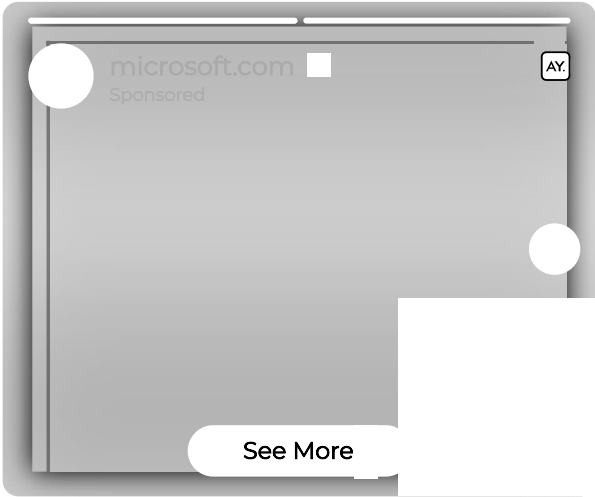
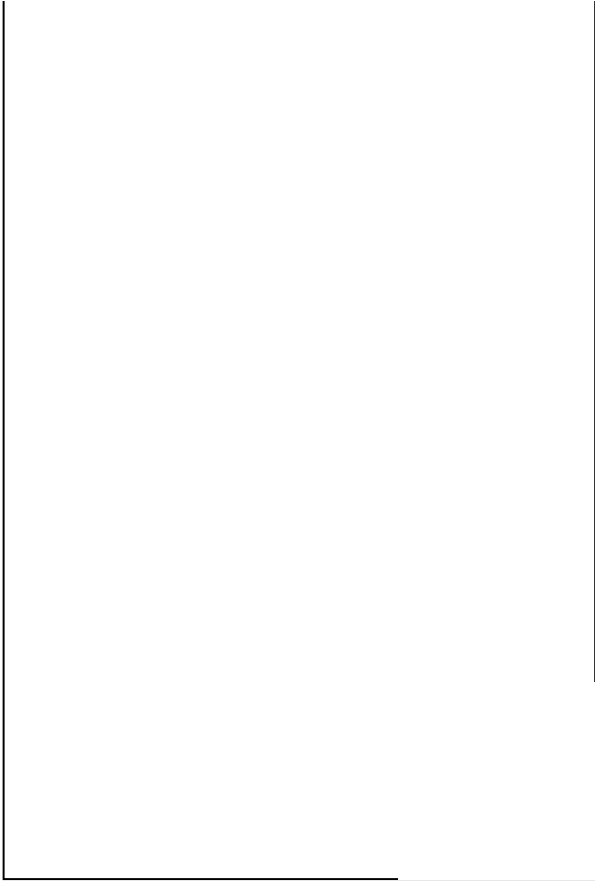


☐ Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment







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](#)



Copyright © 2022 · Hosted on [Cloudways](#) · [Sitemap](#)