IdentityHashMap class in Java

Difficulty Level : Basic Last Updated : 25 Aug, 2021

The **IdentityHashMap** implements Map interface using Hashtable, using reference-equality in place of object-equality when comparing keys (and values). This class is not a general-purpose Map implementation. While this class implements the Map interface, it intentionally violates Map's general contract, which mandates the use of the equals() method when comparing objects. This class is used when the user requires the objects to be compared via reference. It belongs to **java.util** package.

Features of IdentityHashMap

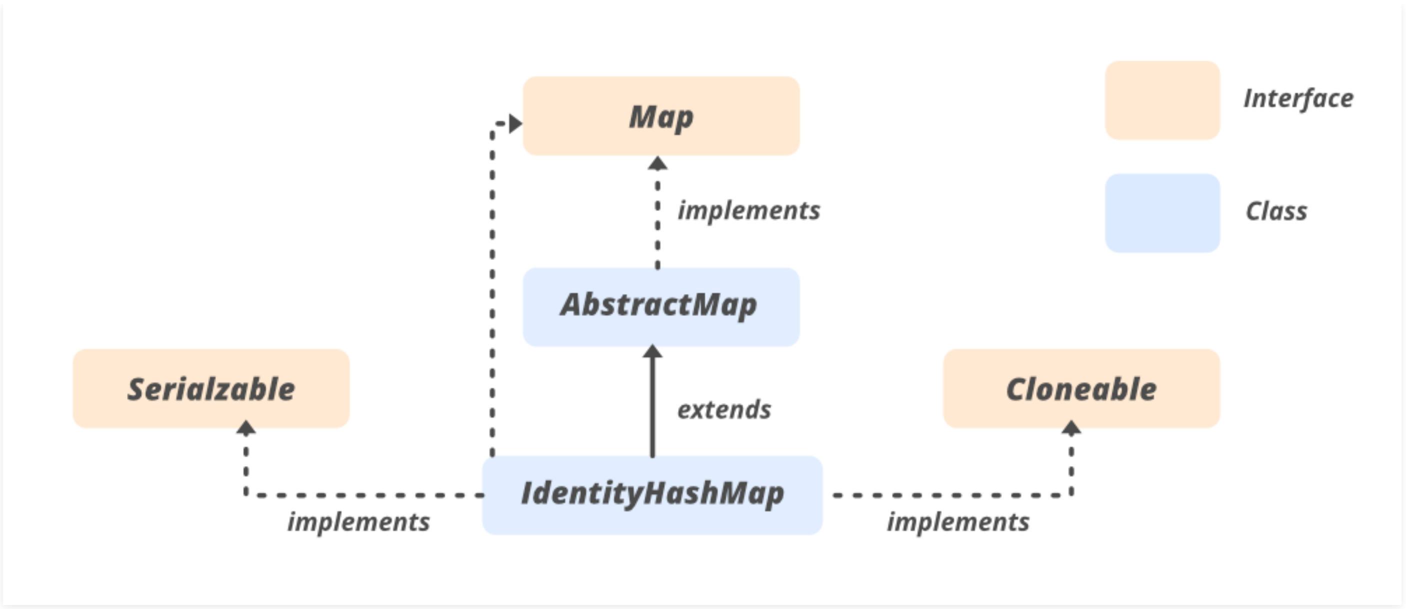
- It follows reference equality, instead of using the equals() method it uses the == operator.
- It is not synchronized and must be synchronized externally.
- Iterators are fail-fast, throw **ConcurrentModificationException** in an attempt to modify while iterating.
- This class provides constant-time performance for the basic operations (get and put), assuming the system identity hash function (System.identityHashCode(Object)) disperses elements properly among the buckets. IdentityHashMap doesn't use hashCode() method instead it uses System.identityHashCode() method. This is a significant difference because now you can use mutable objects as key in Map whose hash code is likely to change when the mapping is stored inside IdentityHashMap.

Declaration:

public class IdentityHashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Serializable, Cloneable

Here, **K** is the key Object type and **V** is the value Object type.

The Hierarchy of IdentityHashMap



It implements **Serializable**, **Cloneable**, Map<K, V> interfaces and extends **AbstractMap<K, V>** class.

Example:

```java
// Java code to demonstrate IdentityHashMap

import java.util.Map;
import java.util.HashMap;
import java.util.IdentityHashMap;

public class IdentityHashMapExample
{
    public static void main(String[] args)
    {
        // creating an instance of IdentityHashMap
        Map<String, String> ihm = new IdentityHashMap<>();

        // Putting key and value pair
        // in a IdentityHashMap Object
        ihm.put("ihmkey","ihmvalue");
        ihm.put(new String("ihmkey"),"ihmvalue1");

        // ihm.size() will print 2 since it
        // compares the objects by reference
        System.out.println("Size of IdentityHashMap--"+ihm.size());

    }
}
```

Output

```
Size of IdentityHashMap--2
```

Constructors of IdentityHashMap

We can create an instance of **IdentityHashMap** in two ways:

```
IdentityHashMap<K, V> ihm = new IdentityHashMap<K, V>();
            (or)
Map<K, V> hm = new IdentityHashMap<K, V>();
```

1. IdentityHashMap(): Constructs a new, empty identity hash map with a default expected maximum size.

IdentityHashMap<K, V> ihm = new IdentityHashMap<K, V>();

2. IdentityHashMap(int expectedMaxSize): Constructs a new, empty map with the specified expected maximum size.

IdentityHashMap<K, V> ihm = new IdentityHashMap(int expectedMaxSize);

3. IdentityHashMap(Map m): Constructs a new identity hash map containing the key-value mappings in the specified map.

IdentityHashMap<K, V> ihm = new IdentityHashMap(Map m);

Basic Operations on IdentityHashMap

1. Adding Elements

To insert or add mapping into an IdentityHashMap, we have put() and putAll() methods. put() can insert a specific key and the value it is mapping, into a particular map. If an existing key is passed then the previous value gets replaced by the new value. putAll() copies all of the elements i.e., the mappings, from one map into another.

```
// Java code to illustrate
// adding elements to IdentityHashMap
import java.util.*;

public class AddingElementsToIdentityHashMap {

    public static void main(String[] args)
    {
        // Creating an empty IdentityHashMap
        Map<Integer, String> identity_hash
            = new IdentityHashMap<Integer, String>();

        // Mapping string values to int keys
        // using put() method
        identity_hash.put(10, "Geeks");
        identity_hash.put(15, "4");
        identity_hash.put(20, "Geeks");
        identity_hash.put(25, "Welcomes");
        identity_hash.put(30, "You");

        // Displaying the IdentityHashMap
        System.out.println("Initial Mappings are: "
                           + identity_hash);

        // Inserting existing key along with new value
          // previous value gets returned and stored in
          // returned_value
        String returned_value
            = (String)identity_hash.put(20, "All");

        // Verifying the returned value
        System.out.println("Returned value is: "
                           + returned_value);

        // Displaying the new map
        System.out.println("New map is: " + identity_hash);

        // Creating a new Identityhash map and copying
        Map<Integer, String> new_Identityhash_map
            = new IdentityHashMap<Integer, String>();
        new_Identityhash_map.putAll(identity_hash);

        // Displaying the final IdentityHashMap
        System.out.println("The new map: "
                               + new_Identityhash_map);
    }
}
```

Output

```
Initial Mappings are: {10=Geeks, 25=Welcomes, 30=You, 20=Geeks, 15=4}
Returned value is: Geeks
New map is: {10=Geeks, 25=Welcomes, 30=You, 20=All, 15=4}
The new map: {10=Geeks, 25=Welcomes, 30=You, 20=All, 15=4}
```

2. Removing Elements

To remove mappings, we use remove(), an inbuilt method of IdentityHashMap class, and is used to remove the mapping of any particular key from the map.

```
// Java code to illustrate removing
// elements from IdentityHashMap

import java.util.*;

public class RemovingMappingsFromIdentityHashMap {
    public static void main(String[] args)
    {

        // Creating an empty IdentityHashMap
        Map<Integer, String> Identity_hash = new
                    IdentityHashMap<Integer, String>();

        // Mapping string values to int keys
        Identity_hash.put(10, "Geeks");
        Identity_hash.put(15, "4");
        Identity_hash.put(20, "Geeks");
        Identity_hash.put(25, "Welcomes");
        Identity_hash.put(30, "You");

        // Displaying the IdentityHashMap
        System.out.println("Initial Mappings are: " +
                                Identity_hash);

        // Removing the existing key mapping
        String returned_value =
                    (String)Identity_hash.remove(20);
        // Verifying the returned value
        System.out.println("Returned value is: " +
                                   returned_value);

        // Displaying the new map
        System.out.println("New map is: " + Identity_hash);
    }
}
```

Output

```
Initial Mappings are: {10=Geeks, 25=Welcomes, 30=You, 20=Geeks, 15=4}
Returned value is: Geeks
New map is: {10=Geeks, 25=Welcomes, 30=You, 15=4}
```

3. Accessing the Elements

We can access the elements of an IdentityHashMap using the get() method, the example of this is given below.

```
// Java code to illustrate the accessing
// elements from IdentityHashMap

import java.util.*;

public class AccessingElementsFromIdentityHashMap {

    public static void main(String[] args)
    {

        // Creating an empty IdentityHashMap
        Map<Integer, String> identity_hash
            = new IdentityHashMap<Integer, String>();

        // Mapping string values to int keys
        identity_hash.put(10, "Geeks");
        identity_hash.put(15, "4");
        identity_hash.put(20, "Geeks");
        identity_hash.put(25, "Welcomes");
        identity_hash.put(30, "You");

        // Displaying the IdentityHashMap
        System.out.println("Initial Mappings are: "
                           + identity_hash);

        // Getting the value of 25
        System.out.println("The Value is: "
                           + identity_hash.get(25));

        // Getting the value of 10
        System.out.println("The Value is: "
                           + identity_hash.get(10));

          // Using keySet() to get the set view of keys
        System.out.println("The set is: " + identity_hash.keySet());

          // Using entrySet() to get the set view
        System.out.println("The set is: " +
                                identity_hash.entrySet());
    }
}
```

Output

```
Initial Mappings are: {10=Geeks, 25=Welcomes, 30=You, 20=Geeks, 15=4}
The Value is: Welcomes
The Value is: Geeks
The set is: [10, 25, 30, 20, 15]
The set is: [10=Geeks, 25=Welcomes, 30=You, 20=Geeks, 15=4]
```

4. Traversing

We can use the Iterator interface to traverse over any structure of the Collection Framework. Since Iterators work with one type of data we use Entry< ? , ? > to resolve the two separate types into a compatible format. Then using the next() method we print the elements of the IdentityHashMap.

```
// Java code to illustrate the
// iterating over IdentityHashmap

import java.util.*;

public class IteratingIdentityHashMap {

    public static void main(String[] args)
    {

        // Creating an empty IdentityHashMap
        IdentityHashMap<Integer, String> identity_hash
            = new IdentityHashMap<Integer, String>();

        // Mapping string values to int keys
        identity_hash.put(10, "Geeks");
        identity_hash.put(15, "4");
        identity_hash.put(20, "Geeks");
        identity_hash.put(25, "Welcomes");
        identity_hash.put(30, "You");

        // Displaying the IdentityHashMap
        System.out.println("Initial Mappings are: "
                           + identity_hash);

        // Create an Iterator over the
        // IdentityHashMap
        Iterator<IdentityHashMap.Entry<Integer, String> >
            itr = identity_hash.entrySet().iterator();

        // The hasNext() method is used to check if there is
        // a next element The next() method is used to
        // retrieve the next element
        while (itr.hasNext()) {
            IdentityHashMap.Entry<Integer, String> entry
                = itr.next();
            System.out.println("Key = " + entry.getKey()
                               + ", Value = "
                               + entry.getValue());
        }
    }
}
```

Output

```
Initial Mappings are: {10=Geeks, 25=Welcomes, 30=You, 20=Geeks, 15=4}
Key = 10, Value = Geeks
Key = 25, Value = Welcomes
Key = 30, Value = You
Key = 20, Value = Geeks
Key = 15, Value = 4
```

Synchronized IdentityHashMap

If multiple threads access an identity hash map concurrently, and at least one of the threads modifies the map structurally, it must be synchronized externally. (A structural modification is any operation that adds or deletes one or more mappings; merely changing the value associated with a key that an instance already contains is not a structural modification.) This is typically accomplished by synchronizing on some object that naturally encapsulates the map. If no such object exists, the map should be “wrapped” using the **Collections.synchronizedMap** method. This is best done at creation time, to prevent accidental unsynchronized access to the map.

Map m = Collections.synchronizedMap(new IdentityHashMap(...));

Methods of IdentityHashMap

- **K** – The type of the keys in the map.
- **V** – The type of values mapped in the map.

METHOD	DESCRIPTION
clear()	Removes all of the mappings from this map.
clone()	Returns a shallow copy of this identity hash map: the keys and values themselves are not cloned.
containsKey(Object key)	Tests whether the specified object reference is a key in this identity hash map.
containsValue(Object value)	Tests whether the specified object reference is a value in this identity hash map.
entrySet()	Returns a Set view of the mappings contained in this map.
equals(Object o)	Compares the specified object with this map for equality.
get(Object key)	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
hashCode()	Returns the hash code value for this map.
isEmpty()	Returns true if this identity hash map contains no key-value mappings.
keySet()	Returns an identity-based set view of the keys contained in this map.

METHOD	DESCRIPTION
put(K key, V value)	Associates the specified value with the specified key in this identity hash map.
putAll(Map<? extends K,? extends V> m)	Copies all of the mappings from the specified map to this map.
remove(Object key)	Removes the mapping for this key from this map if present.
size()	Returns the number of key-value mappings in this identity hash map.
values()	Returns a Collection view of the values contained in this map.

Methods declared in class java.util.AbstractMap

METHOD	DESCRIPTION
toString()	Returns a string representation of this map.

Methods declared in interface java.util.Map

METHOD	DESCRIPTION
compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	Attempts to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)	If the specified key is not already associated with a value (or is mapped to null), attempts to compute its value using the given mapping function and enters it into this map unless null.
computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	If the value for the specified key is present and non-null, attempts to compute a new mapping given the key and its current mapped value.
forEach(BiConsumer<? super K,? super V> action)	Performs the given action for each entry in this map until all entries have been processed or the action throws an exception.

METHOD	DESCRIPTION
getOrDefault(Object key, V defaultValue)	Returns the value to which the specified key is mapped, or defaultValue if this map contains no mapping for the key.
merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)	If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.
putIfAbsent(K key, V value)	If the specified key is not already associated with a value (or is mapped to null) associates it with the given value and returns null, else returns the current value.
remove(Object key, Object value)	Removes the entry for the specified key only if it is currently mapped to the specified value.
replace(K key, V value)	Replaces the entry for the specified key only if it is currently mapped to some value.
replace(K key, V oldValue, V newValue)	Replaces the entry for the specified key only if currently mapped to the specified value.
replaceAll(BiFunction<? super K,? super V,? extends V> function)	Replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.

IdentityHashMap vs HashMap

- IdentityHashMap uses the equality operator "==" for comparing keys and values while HashMap uses the equals method for comparing keys and values inside Map.
- Since IdentityHashMap doesn't use equals() its comparatively faster than HashMap for an object with expensive equals().
- IdentityHashMap doesn't require keys to be immutable as it is not relied on equals().

The below program illustrates the difference between IdentityHashMap and HashMap implementation.

```
// Java code to demonstrate IdentityHashMap and
// illustration of how it is different from HashMap

import java.util.Map;
import java.util.HashMap;
import java.util.IdentityHashMap;

public class IdentityHashMapExample
{
    public static void main(String[] args)
    {
        // Creating HashMap and IdentityHashMap objects
        Map<String, String> hm = new HashMap<>();
        Map<String, String> ihm = new IdentityHashMap<>();

        // Putting key and value in HashMap and IdentityHashMap Object
        hm.put("hmkey","hmvalue");
        hm.put(new String("hmkey"),"hmvalue1");
        ihm.put("ihmkey","ihmvalue");
        ihm.put(new String("ihmkey"),"ihmvalue1");

        // Print Size of HashMap and WeakHashMap Object
        // hm.size() will print 1 since it compares the objects logically
        // and both the keys are same
        System.out.println("Size of HashMap is : "+hm.size());

        // ihm.size() will print 2 since it compares the objects by reference
        System.out.println("Size of IdentityHashMap is : "+ihm.size());


    }
}
```

Output

```
Size of HashMap is : 1
Size of IdentityHashMap is : 2
```

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.