CSCI 2120:
Software Design & Development II

UNIT 2: Collections Framework & Generics

HashMap

Overview

1. Introduction
2. Hierarchy of HashMap class
3. HashMap class Declaration
4. Features of HashMap class
5. Constructors of HashMap class
6. HashMap methods
7. HashMap Examples
8. Use of HashMap

Introduction

- **HashMap in Java** is an unordered collection that stores elements (objects) in the form of key-value pairs (called entries).
- It is expressed as `HashMap<Key, Value>`, or `HashMap<K, V>`, where K stands for key and V for value. Both Key and Value are objects. `HashMap` uses an object to retrieve another object.
- If the key is provided, its associated value can be easily retrieved from the `HashMap`. Keys in the map must be unique which means we cannot use duplicate data for keys in the `HashMap`.
- If we try to insert an entry that has a duplicate key, the map replaces the old entry with a new entry.
- `HashMap` class is one of four concrete implementations of the `Map` interface. It was added in Java 1.2 version. It is found in `java.util.HashMap<K, V>` package. It is efficient for locating a value, adding an entry, and deleting an entry.

Hierarchy of HashMap class

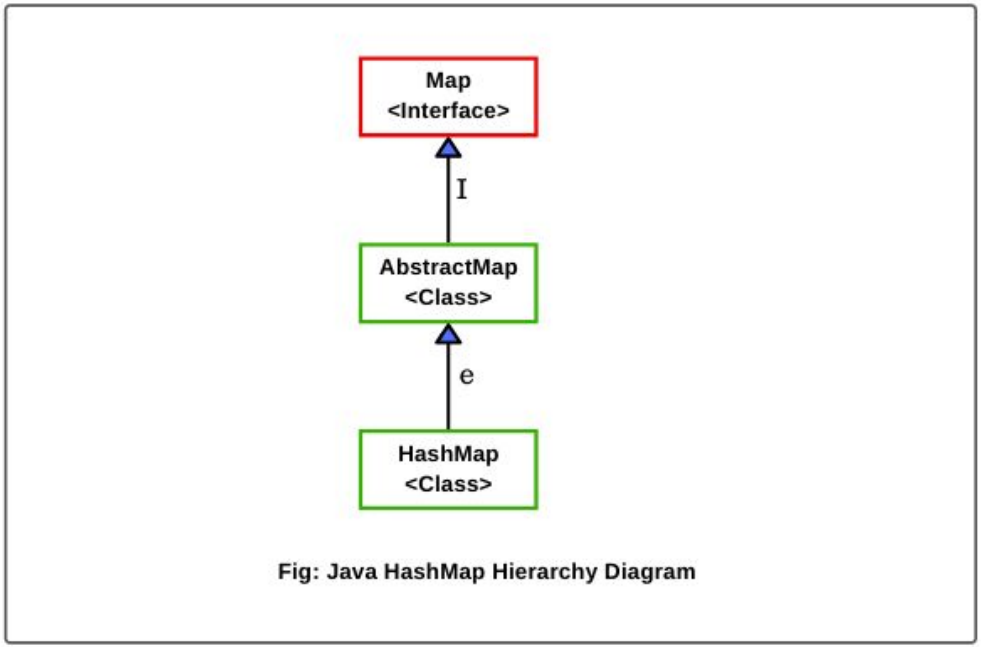

Fig: Java HashMap Hierarchy Diagram

`HashMap` class extends `AbstractMap` class and implements the `Map` interface.

The hierarchy diagram of `HashMap` can be seen in the figure.

HashMap class Declaration

HashMap is a generic class that can be declared as follows:

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

Features of Java HashMap class

There are several features of `HashMap` class that need to keep in mind. They are as:

1. The underlying data structure of `HashMap` is HashTable. In simple words, `HashMap` internally uses hash table for storing entries. That means accessing and adding an entry almost as fast as accessing an array.

2. In `HashMap`, insertion order is not preserved (i.e. maintains no order). Which means we cannot retrieve keys and values in the same order in which they have been inserted into the `HashMap`.

3. It is based on the Hashcode of keys, not on the hash code of values.

4. `HashMap` contains only unique keys that means no duplicate keys are allowed but values can be duplicated. We retrieve values based on the key.

5. Heterogeneous objects are allowed for both keys and values.

6. `HashMap` can have only one null key because duplicate keys are not allowed.

Features of Java HashMap class

There are several features of HashMap class that need to keep in mind. They are as:

7. Multiple null values are allowed in the `HashMap`.

8. `HashMap` is not synchronized that means while using multiple threads on the `HashMap` object, we will get unreliable results.

9. `HashMap` implements `Cloneable`, and `Serializable` interfaces but not `RandomAccess`.

10. `HashMap` is the best choice if our frequent operation is a search operation.

11. If no element exists in the HashMap, it will throw an exception named **NoSuchElementException**.

12. `HashMap` stores only object references. Therefore, we cannot use primitive data types like `double` or `int`. We can use wrapper class like `Integer` or `Double` instead.

Constructors of HashMap class

HashMap class in Java provides four constructors that are as follows:

Constructors of HashMap class

1. HashMap(): It is used to construct an empty HashMap object with the default initial capacity of 16 and the default fill ratio (load factor) is 0.75.

The syntax to create a HashMap object is as follows:

```
HashMap hmap = new HashMap();

// Generic form
HashMap<K, V> hmap = new HashMap<K,V>();
```

Constructors of HashMap class

2. HashMap(int initialCapacity): It is used to create an empty hashmap object with a specified initial capacity under the default load factor 0.75.

The general syntax is as follows:

```
HashMap<K,V> hmap = new HashMap<K,V>(int initialCapacity);
```

Constructors of HashMap class

3. HashMap(Map m): This constructor is used to create `HashMap` object by initializing the elements of the given Map object m.

The general syntax is as follows:

```
HashMap<K,V> hmap = new HashMap<K,V>(Map m);
```

Constructors of HashMap class

4. HashMap(int initialCapacity, float loadFactor): This constructor is used to create `HashMap` object with specified initial capacity and load factor.

The general syntax to create `HashMap` object with initial capacity and load factor is as:

```
HashMap<K,V> hmap = new HashMap<>(int initialCapacity, float loadFactor);

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

- The capacity is the number of buckets in which hash map values can be stored. The load factor is the amount of buckets that are used before the capacity automatically is grown.

- The value is a floating-point number that has ranges from 0 (empty) to 1.0 (full). 0.7 means when buckets are 70% full, the capacity is increased. The default capacity is 16 and the load factor is 0.75 that is often sufficient.

HashMap methods

Method	Description
void clear()	It is used to remove entries from the specified map.
boolean isEmpty()	This method is used to check whether the map is empty or not. If there are no key-value mapping present in the hash map then it returns true, else false.
Object clone()	It is used to create a shallow copy of this HashMap. Only hashmap and all entries are cloned, not elements. Both this map and new map share references to the same keys and values.
Set entrySet()	It is used to return a set view containing all of the key/value pairs in this map.
Set keySet()	This method is used to retrieve a set view of the keys in this map.
V put(Object key, Object value)	This method is used to insert an entry in the map.

HashMap methods

Method	Description
void putAll(Map map)	This method is used to insert all the entries of the specified map to another map.
V putIfAbsent(K key, V value)	This method adds the specified value associated with the specified key in the map only if it is not already specified.
V remove(Object key)	This method is used to delete an entry for the specified key.
boolean remove(Object key, Object value)	This method removes the specified value associated with specific key from the map.
int size()	This method returns the number of entries in the map.
Collection values()	This method returns a collection view containing all of the values in the map.

HashMap methods

Method	Description
V get(Object key)	This method is used to retrieve the value associated with a key. Its return type is Object.
V replace(K key, V value)	This method is used to replace the specified value for a specified key.
boolean replace(K key, V oldV, V newV)	This method is used to replace the old value with the new value for a specified key.
boolean containsValue(Object v)	This method is used to determine if map contains a particular value. It returns true if some value is equal to the v
boolean containsKey(Object k)	This method is used to determine if the map contains a particular key. It will return true if some key in this map is equal to k.
boolean equals(Object o)	This method is used to compare the specified Object with the Map.

HashMap Examples

Example 1: Adding entries

1. Let’s create a program where we will simply add entries in the HashMap and display it on the console.

Example 1: Adding entries

```
import java.util.HashMap;
public class HashMapTester1 {
   public static void main(String[] args) {
       // Create a HashMap.
       HashMap<String,Integer> hmap = new HashMap<>();

       // Checking HashMap is empty or not.
       boolean empty = hmap.isEmpty();
       System.out.println("Is HashMap empty: " +empty);

       // Adding entries in the hash map.
       hmap.put("John", 24); // hmap.size() is 1.
       hmap.put("Deep", 22); // hmap.size() is 2.
       hmap.put("Shubh", 15); // hmap.size() is 3.
       hmap.put("Riky", 22); // hmap.size() is 4. // Adding duplicate value.
       hmap.put("Mark", 30); // hmap.size() is 5.

       System.out.println("Entries in HashMap: " +hmap);
       int size = hmap.size();
       System.out.println("Size of HashMap: " +size);

       // Adding null key and value.
       hmap.put(null, null); // hmap.size() is 6.
       System.out.println("Updated entries in HashMap: " +hmap);
   }
}
```

Example 1: Adding entries

Output:

```
Is HashMap empty: true
Entries in HashMap: {Riky=22, Shubh=15, John=24, Mark=30, Deep=22}
Size of HashMap: 5
Updated entries in HashMap: {null=null, Riky=22, Shubh=15, John=24, Mark=30, Deep=22}
```

As you can see from the above output, the entries in the `HashMap` are in no particular order in which they are inserted in map. We have stored `String` as keys, and `Integer` as values as so we are using `HashMap<String, Integer>` as the type. The `put()` method adds the entries to the map.

Example 2: Try adding duplicate entries

2. Let’s take an example program in which we will try to add duplicate keys and values in HashMap.

Example 2: Try adding duplicate entries

```
import java.util.HashMap;
public class HashMapTester2 {
   public static void main(String[] args) {
       HashMap<Integer, String> hmap = new HashMap<>();

       hmap.put(5, "Banana");
       hmap.put(10, "Mango");
       hmap.put(15, "Apple");

       System.out.println("Entries in HashMap: " +hmap);
       System.out.println("Size of HashMap: " +hmap.size());

       // Adding duplicate key in hash map.
       hmap.put(10, "Guava"); // Still hmap.size is 3.
       hmap.put(20, "Banana"); // Adding duplicate value.

       System.out.println("Updated entries in HashMap: " +hmap);
       System.out.println("Size after adding duplicate value: " +hmap.size());
   }
}
```

Example 2: Try adding duplicate entries

Output:

```
Entries in HashMap: {5=Banana, 10=Mango, 15=Apple}
Size of HashMap: 3
Updated entries in HashMap: {20=Banana, 5=Banana, 10=Guava, 15=Apple}
Size after adding duplicate value: 4
```

As you can see in this program, we cannot store duplicate keys in `HashMap`. However, we have tried to store a duplicate key with another value, but it simply replaced the value.

Example 3: Removing entries

3. Let’s create a program where we will perform the remove operation. We will remove entry from the HashMap using remove() method.

Example 3: Removing entries

```
import java.util.HashMap;
public class HashMapTester3 {
   public static void main(String[] args) {
       HashMap<Character,String> hmap = new HashMap<>();

       hmap.put('R', "Red");
       hmap.put('O', "Orange");
       hmap.put('G', "Green");
       hmap.put('B', "Brown");
       hmap.put('W', "White");

       // Displaying HashMap entries.
       System.out.println("Entries in HashMap: " +hmap);

       // Removing Key-Value pairs for key 'B'.
       Object removeEntry = hmap.remove('B');
       System.out.println("Removed Entry: " +removeEntry);
       System.out.println("HashMap Entries after remove: " +hmap);

       // Checking entry is removed or not.
       Object removeEntry2 = hmap.remove('W', "White");
       System.out.println("Is entry removed: " +removeEntry2);
       System.out.println("Updated HashMap entries: " +hmap);
   }
}
```

Example 3: Removing entries

Output:

```
Entries in HashMap: {R=Red, B=Brown, G=Green, W=White, O=Orange}
Removed Entry: Brown
HashMap Entries after remove: {R=Red, G=Green, W=White, O=Orange}
Is entry removed: true
Updated HashMap entries: {R=Red, G=Green, O=Orange}
```

Example 4: Replacing values

4. Let’s take an example program where we will replace a specified value for the specified key.

Example 4: Replacing values

```
import java.util.HashMap;
public class HashMapTester4 {
   public static void main(String[] args) {
       HashMap<Character,String> hmap = new HashMap<>();

       hmap.put('R', "Red");
       hmap.put('O', "Orange");
       hmap.put('G', "Green");
       hmap.put('B', "Brown");
       hmap.put('W', "White");

       // Displaying HashMap entries.
       System.out.println("Entries in HashMap: " +hmap);

       // Replacing specified value for the specified key.
       Object replaceValue = hmap.replace('B', "Black");
       System.out.println("Replaced value: " +replaceValue);
       System.out.println("Updated entries in HashMap: " +hmap);

       boolean replaceValue2 = hmap.replace('G', "Green", "Greenish");
       System.out.println("Is value replaced: " +replaceValue2);
       System.out.println(hmap);
   }
}
```

Example 4: Replacing values

Output:

```
Entries in HashMap: {R=Red, B=Brown, G=Green, W=White, O=Orange}
Replaced value: Brown
Updated entries in HashMap: {R=Red, B=Black, G=Green, W=White, O=Orange}
Is value replaced: true
{R=Red, B=Black, G=Greenish, W=White, O=Orange}
```

Use of HashMap

`HashMap` can be the best choice if we want to perform a search operation. It is designed to rapidly find things.

The best example of this kind is phonebook. The name of person (a string) can be used to search the person's phone number.

Let's understand it with a simple example program. Look at the following source code.

Example 5: Searching entries

```
import java.util.HashMap;
public class HashMapTester5 {
   public static void main(String[] args) {
       HashMap<String, Long> hmap = new HashMap<>();

       hmap.put("John", 9431676282L);
       hmap.put("Deep", 8292736478L);
       hmap.put("Shubh", 8123543268L);
       hmap.put("Mark", 9876789142L);
       hmap.put("Ricky", 8768976872L);

       // Retrieve number with its key by calling get() method.
       Long number = hmap.get("Deep");
       System.out.println("Deep's phone number: " +number);

       Long number2 = hmap.getOrDefault("Steave", -1L);
       System.out.println("Alex's phone number: " +number2);
   }
}
```

Example 5: Searching entries

Output:

```
Deep's phone number: 8292736478
Alex's phone number: -1
```

In this example, we have created a `HashMap` called phonebook with keys (person's name) that are `String` and values (person's phone number) that are `Long` objects. Objects are stored in the `HashMap` by calling `put(Object key, Object value)` method.

This method stores an item on the map with key as name and value as a phone number. `hmap.put("John", 9431676282L);`. If the specified key is not found, `-1` is returned by default.

END