Java Hashtable class

Java Hashtable class implements a hashtable, which maps keys to values. It inherits Dictionary class and implements the Map interface.

Points to remember

- A Hashtable is an array of a list. Each list is known as a bucket. The position of the bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.
- Java Hashtable class contains unique elements.
- Java Hashtable class doesn't allow null key or value.
- Java Hashtable class is synchronized.
- The initial default capacity of Hashtable class is 11 whereas loadFactor is 0.75.

Hashtable class declaration

Let's see the declaration for java.util.Hashtable class.

public class Hashtable<K,V> extends Dictionary<K,V> implements Map<K,V>, Cloneable, Serializable

Hashtable class Parameters

Let's see the Parameters for java.util.Hashtable class.

- K: It is the type of keys maintained by this map.
- V: It is the type of mapped values.

Constructors of Java Hashtable class

Constructor	Description
Hashtable()	It creates an empty hashtable having the initial default capacity and load factor.
Hashtable(int capacity)	It accepts an integer parameter and creates a hash table that contains a specified initial capacity.
Hashtable(int capacity, float loadFactor)	It is used to create a hash table having the specified initial capacity and loadFactor.

Hashtable(Map<? extends K,? extends V> t)	It creates a new hash table with the same mappings as the given Map.

Methods of Java Hashtable class

Method	Description
void clear()	It is used to reset the hash table.
Object clone()	It returns a shallow copy of the Hashtable.
V compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	It is used to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
V computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)	It is used to compute its value using the given mapping function, if the specified key is not already associated with a value (or is mapped to null), and enters it into this map unless null.
V computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	It is used to compute a new mapping given the key and its current mapped value if the value for the specified key is present and non-null.
Enumeration elements()	It returns an enumeration of the values in the hash table.
Set<Map.Entry<K,V>> entrySet()	It returns a set view of the mappings contained in the map.
boolean equals(Object o)	It is used to compare the specified Object with the Map.
void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.
V getOrDefault(Object key, V defaultValue)	It returns the value to which the specified key is mapped, or defaultValue if the map contains no mapping for the key.
int hashCode()	It returns the hash code value for the Map
Enumeration<K> keys()	It returns an enumeration of the keys in the hashtable.
Set<K> keySet()	It returns a Set view of the keys contained in the map.
V merge(K key, V value, BiFunction<? super V,? super V,? extends V>	If the specified key is not already associated with a value or is associated with null, associates it with the given non-null

remappingFunction)	value.
V put(K key, V value)	It inserts the specified value with the specified key in the hash table.
void putAll(Map<? extends K,? extends V> t))	It is used to copy all the key-value pair from map to hashtable.
V 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.
boolean remove(Object key, Object value)	It removes the specified values with the associated specified keys from the hashtable.
V replace(K key, V value)	It replaces the specified value for a specified key.
boolean replace(K key, V oldValue, V newValue)	It replaces the old value with the new value for a specified key.
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It 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.
String toString()	It returns a string representation of the Hashtable object.
Collection values()	It returns a collection view of the values contained in the map.
boolean contains(Object value)	This method returns true if some value equal to the value exists within the hash table, else return false.
boolean containsValue(Object value)	This method returns true if some value equal to the value exists within the hash table, else return false.
boolean containsKey(Object key)	This method return true if some key equal to the key exists within the hash table, else return false.
boolean isEmpty()	This method returns true if the hash table is empty; returns false if it contains at least one key.
protected void rehash()	It is used to increase the size of the hash table and rehashes all of its keys.
V get(Object key)	This method returns the object that contains the value associated with the key.
V remove(Object key)	It is used to remove the key and its value. This method

	returns the value associated with the key.
int size()	This method returns the number of entries in the hash table.

Java Hashtable Example

```
import java.util.*;
class Hashtable1{
 public static void main(String args[]){
  Hashtable<Integer,String> hm=new Hashtable<Integer,String>();

  hm.put(100,"Amit");
  hm.put(102,"Ravi");
  hm.put(101,"Vijay");
  hm.put(103,"Rahul");

  for(Map.Entry m:hm.entrySet()){
   System.out.println(m.getKey()+" "+m.getValue());
  }
 }
}
```

Output:

```
103 Rahul
102 Ravi
101 Vijay
100 Amit
```

Java Hashtable Example: remove()

```
import java.util.*;
public class Hashtable2 {
   public static void main(String args[]) {
  Hashtable<Integer,String> map=new Hashtable<Integer,String>();
    map.put(100,"Amit");
    map.put(102,"Ravi");
    map.put(101,"Vijay");
    map.put(103,"Rahul");
    System.out.println("Before remove: "+ map);
      // Remove value for key 102
      map.remove(102);
      System.out.println("After remove: "+ map);
   }
}
```

Output:

Before remove: {103=Rahul, 102=Ravi, 101=Vijay, 100=Amit}
After remove: {103=Rahul, 101=Vijay, 100=Amit}

Java Hashtable Example: getOrDefault()

```
import java.util.*;
class Hashtable3{
 public static void main(String args[]){
   Hashtable<Integer,String> map=new Hashtable<Integer,String>();
    map.put(100,"Amit");
    map.put(102,"Ravi");
    map.put(101,"Vijay");
    map.put(103,"Rahul");
    //Here, we specify the if and else statement as arguments of the method
    System.out.println(map.getOrDefault(101, "Not Found"));
    System.out.println(map.getOrDefault(105, "Not Found"));
 }
}
```

Output:
Vijay
Not Found

Java Hashtable Example: putIfAbsent()

```
import java.util.*;
class Hashtable4{
 public static void main(String args[]){
   Hashtable<Integer,String> map=new Hashtable<Integer,String>();
    map.put(100,"Amit");
    map.put(102,"Ravi");
    map.put(101,"Vijay");
    map.put(103,"Rahul");
    System.out.println("Initial Map: "+map);
    //Inserts, as the specified pair is unique
    map.putIfAbsent(104,"Gaurav");
    System.out.println("Updated Map: "+map);
    //Returns the current value, as the specified pair already exist
    map.putIfAbsent(101,"Vijay");
    System.out.println("Updated Map: "+map);
 }
}
```

Output:
Initial Map: {103=Rahul, 102=Ravi, 101=Vijay, 100=Amit}
Updated Map: {104=Gaurav, 103=Rahul, 102=Ravi, 101=Vijay, 100=Amit}
Updated Map: {104=Gaurav, 103=Rahul, 102=Ravi, 101=Vijay, 100=Amit}

Java Hashtable Example: Book

```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
   this.id = id;
   this.name = name;
   this.author = author;
   this.publisher = publisher;
   this.quantity = quantity;
}
}
public class HashtableExample {
public static void main(String[] args) {
   //Creating map of Books
   Map<Integer,Book> map=new Hashtable<Integer,Book>();
   //Creating Books
   Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
   Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);
   Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
   //Adding Books to map
   map.put(1,b1);
   map.put(2,b2);
   map.put(3,b3);
   //Traversing map
   for(Map.Entry<Integer, Book> entry:map.entrySet()){
      int key=entry.getKey();
      Book b=entry.getValue();
      System.out.println(key+" Details:");
      System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
   }
}
}
```

Output:

```
3 Details:
103 Operating System Galvin Wiley 6
2 Details:
102 Data Communications & Networking Forouzan Mc Graw Hill 4
1 Details:
101 Let us C Yashwant Kanetkar BPB 8
```