JAVA - THE HASHTABLE CLASS

Hashtable was part of the original java.util and is a concrete implementation of a Dictionary.

However, Java 2 re-engineered Hashtable so that it also implements the Map interface. Thus, Hashtable is now integrated into the collections framework. It is similar to HashMap, but is synchronized.

Like HashMap, Hashtable stores key/value pairs in a hash table. When using a Hashtable, you specify an object that is used as a key, and the value that you want linked to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.

Below given is the list of constructors provided by the HashTable class.

Sr.No	Constructor and Description
1	**Hashtable** This is the default constructor of the hash table it instantiates the Hashtable class.
2	**Hashtable*intsize*** This constructor accepts an integer parameter and creates a hash table that has an initial size specified by integer value size.
3	**Hashtable*intsize, floatfillRatio*** This creates a hash table that has an initial size specified by size and a fill ratio specified by fillRatio. This ratio must be between 0.0 and 1.0, and it determines how full the hash table can be before it is resized upward.
4	**Hashtable*intsize, floatfillRatio*** This constructor creates a hash table that is initialized with the elements in m. The capacity of the hash table is set to twice the number of elements in m. The default load factor of 0.75 is used.

Apart from the methods defined by Map interface, Hashtable defines the following methods:

Sr.No	Methods with Description
1	**void clear** Resets and empties the hash table.
2	**Object clone** Returns a duplicate of the invoking object.
3	**boolean contains*Objectvalue*** Returns true if some value equal to value exists within the hash table. Returns false if the value isn't found.

4	**boolean containsKey***Objectkey* Returns true if some key equal to key exists within the hash table. Returns false if the key isn't found.
5	**boolean containsValue***Objectvalue* Returns true if some value equal to value exists within the hash table. Returns false if the value isn't found.
6	**Enumeration elements** Returns an enumeration of the values contained in the hash table.
7	**Object get***Objectkey* Returns the object that contains the value associated with key. If key is not in the hash table, a null object is returned.
8	**boolean isEmpty** Returns true if the hash table is empty; returns false if it contains at least one key.
9	**Enumeration keys** Returns an enumeration of the keys contained in the hash table.
10	**Object put***Objectkey, Objectvalue* Inserts a key and a value into the hash table. Returns null if key isn't already in the hash table; returns the previous value associated with key if key is already in the hash table.
11	**void rehash** Increases the size of the hash table and rehashes all of its keys.
12	**Object remove***Objectkey* Removes key and its value. Returns the value associated with key. If key is not in the hash table, a null object is returned.
13	**int size** Returns the number of entries in the hash table.
14	**String toString** Returns the string equivalent of a hash table.

Example:

The following program illustrates several of the methods supported by this data structure:

```
import java.util.*;

public class HashTableDemo {
```

```
   public static void main(String args[]) {
      // Create a hash map
      Hashtable balance = new Hashtable();
      Enumeration names;
      String str;
      double bal;

      balance.put("Zara", new Double(3434.34));
      balance.put("Mahnaz", new Double(123.22));
      balance.put("Ayan", new Double(1378.00));
      balance.put("Daisy", new Double(99.22));
      balance.put("Qadir", new Double(-19.08));

      // Show all balances in hash table.
      names = balance.keys();
      while(names.hasMoreElements()) {
         str = (String) names.nextElement();
         System.out.println(str + ": " +
         balance.get(str));
      }
      System.out.println();
      // Deposit 1,000 into Zara's account
      bal = ((Double)balance.get("Zara")).doubleValue();
      balance.put("Zara", new Double(bal+1000));
      System.out.println("Zara's new balance: " +
      balance.get("Zara"));
   }
}
```

This would produce the following result:

```
Qadir: -19.08
Zara: 3434.34
ahnaz: 123.22
Daisy: 99.22
Ayan: 1378.0

Zara's new balance: 4434.34
```