Set Custom implementation - HashSet - add, contains, remove Employee object in java

You are here : Home / Core Java Tutorials / Data structures / Collection framework

Contents of page :

In this post i will be explaining how to **put, get, remove Employee object in custom** HashSet.

1) Understanding equals and hascode method of Employee class in java >

Employee object overrides:
>**equals** method - helps in checking equality of employee objects used as key in entry objects.
>**hashCode** method - helps in finding bucket's index on which data will be stored.

```
    @Override
    public boolean equals(Object o){

         if(o==null)
                return false;
         if(this.getClass()!=o.getClass())
                return false;

         Employee e=(Employee)o;
         return e.id.equals(this.id) && e.name.equals(this.name);
    }

    @Override
    public int hashCode(){
         return id.hashCode() + name.hashCode();
    }
```

We will maintain **bucket (ArrayList)** which will store **Entry (LinkedList)**.

Must read: Sorted DoublyLinkedList - insert and delete specific Node.

2) Full Program/SourceCode to put, get, remove Employee object in custom HashSet in java>

```
package com.ankit;

 /** Copyright (c), AnkitMittal  JavaMadeSoEasy.com */
/**
* @author AnkitMittal
* Copyright (c), AnkitMittal . All Contents are copyrighted and must not be reproduced in any form.
* This class provides custom implementation of Set(without using java api's- we will be using HashMapCustom)- which allows does not allow you to store duplicate values.
* Note- implementation does not allow you to store null values.
* @param <K>
* @param <V>
*/

class HashSetCustom<E>{

    private HashMapCustom<E, Object> hashMapCustom;

    public HashSetCustom(){
         hashMapCustom=new HashMapCustom<>();
    }

    /**
     * add objects in SetCustom.
     */
    public void add(E value){
         hashMapCustom.put(value, null);

    }

    /**
     * Method returns true if set contains the object.
     * @param key
     */
    public boolean contains(E obj){
         return hashMapCustom.contains(obj) !=null ? true :false;
    }

    /**
     * Method displays all objects in setCustom.
     * insertion order is not guaranteed, for maintaining insertion order refer LinkedHashSet.
     */
    public void display(){
       hashMapCustom.displaySet();
    }

    /**
     * Method removes object from setCustom.
     * insertion order is not guaranteed, for maintaining insertion order refer LinkedHashSet.
     * @param obj
     */
    public boolean remove(E obj){
       return hashMapCustom.remove(obj);
    }


}
 /** Copyright (c), AnkitMittal  JavaMadeSoEasy.com */
/**
* @author AnkitMittal
* Copyright (c), AnkitMittal . All Contents are copyrighted and must not be reproduced in any form.
* Employee class- to be used as key in HashSetCustom.
*/
class Employee {
    private String id;
    private String name;

    /**
     * Employee constructor
     */
    public Employee(String id, String name) { // constructor
         this.id = id;
         this.name = name;
    }

    @Override
    public String toString() {
         return "Employee[id=" + id + ", name=" + name + "] ";
    }


    @Override
    public boolean equals(Object o){

         if(o==null)
                return false;
         if(this.getClass()!=o.getClass())
                return false;

         Employee e=(Employee)o;
         return e.id.equals(this.id) && e.name.equals(this.name);
    }

    @Override
    public int hashCode(){
         return id.hashCode() + name.hashCode();
    }

}




 /**
 * @author AnkitMittal
 * Copyright (c), AnkitMittal . All Contents are copyrighted and must not be reproduced in any form.
 * This class provides custom implementation of HashMap(without using java api's)- which allows us to store data in key-value pair form..
 * @param <K>
 * @param <V>
 */
class HashMapCustom<K, V> {

     private Entry<K,V>[] table;   //Array of Entry.
     private int capacity= 4;  //Initial capacity of HashMap


     static class Entry<K, V> {
         K key;
         V value;
         Entry<K,V> next;

         public Entry(K key, V value, Entry<K,V> next){
             this.key = key;
             this.value = value;
             this.next = next;
         }
     }


    @SuppressWarnings("unchecked")
    public HashMapCustom(){
       table = new Entry[capacity];
    }



    /**
     * Method allows you put key-value pair in HashMapCustom.
     * If the map already contains a mapping for the key, the old value is replaced.
     * Note: method does not allows you to put null key thought it allows null values.
     * Implementation allows you to put custom objects as a key as well.
     * Key Features: implementation provides you with following features:-
     *     >provide complete functionality how to override equals method.
     *  >provide complete functionality how to override hashCode method.
     * @param newKey
     * @param data
     */
    public void put(K newKey, V data){
       if(newKey==null)
           return;    //does not allow to store null.

       int hash=hash(newKey);
       Entry<K,V> newEntry = new Entry<K,V>(newKey, data, null);

        if(table[hash] == null){
         table[hash] = newEntry;
        }else{
           Entry<K,V> previous = null;
           Entry<K,V> current = table[hash];

           while(current != null){ //we have reached last entry of bucket.
           if(current.key.equals(newKey)){
               if(previous==null){  //node has to be insert on first of bucket.
                     newEntry.next=current.next;
                     table[hash]=newEntry;
                     return;
               }
               else{
               newEntry.next=current.next;
               previous.next=newEntry;
               return;
               }
           }
           previous=current;
             current = current.next;
         }
         previous.next = newEntry;
        }
    }

    /**
     * Method returns value corresponding to key.
     * @param key
     */
    public V get(K key){
        int hash = hash(key);
        if(table[hash] == null){
         return null;
        }else{
         Entry<K,V> temp = table[hash];
         while(temp!= null){
             if(temp.key.equals(key))
                 return temp.value;
             temp = temp.next; //return value corresponding to key.
         }
         return null;   //returns null if key is not found.
        }
    }


    /**
     * Method removes key-value pair from HashMapCustom.
     * @param key
     */
    public boolean remove(K deleteKey){

       int hash=hash(deleteKey);

      if(table[hash] == null){
            return false;
      }else{
        Entry<K,V> previous = null;
        Entry<K,V> current = table[hash];

        while(current != null){ //we have reached last entry node of bucket.
           if(current.key.equals(deleteKey)){
               if(previous==null){  //delete first entry node.
                     table[hash]=table[hash].next;
                     return true;
               }
               else{
                     previous.next=current.next;
                      return true;
               }
           }
           previous=current;
             current = current.next;
          }
        return false;
      }

    }


    /**
     * Method displays all key-value pairs present in HashMapCustom.,
     * insertion order is not guaranteed, for maintaining insertion order refer LinkedHashMapCustom.
     * @param key
     */
    public void display(){

       for(int i=0;i<capacity;i++){
           if(table[i]!=null){
                  Entry<K, V> entry=table[i];
                  while(entry!=null){
                        System.out.print("{"+entry.key+"="+entry.value+"}" +" ");
                        entry=entry.next;
                  }
           }
       }

    }



    /**
     * Method returns true if set contains the object.
     * @param key
     */
    public K contains(K key){
        int hash = hash(key);
        if(table[hash] == null){
         return null;
        }else{
         Entry<K,V> temp = table[hash];
         while(temp!= null){
             if(temp.key.equals(key))
                 return key;
             temp = temp.next; //return value corresponding to key.
         }
         return null;   //returns null if key is not found.
        }
    }


    /**
     * Method displays all objects in setCustom.
     * insertion order is not guaranteed, for maintaining insertion order refer LinkedHashSet.
     */
    public void displaySet(){

       for(int i=0;i<capacity;i++){
           if(table[i]!=null){
                  Entry<K, V> entry=table[i];
                  while(entry!=null){
                        System.out.print(entry.key+" ");
                        entry=entry.next;
                  }
           }
       }

    }

    /**
     * Method implements hashing functionality, which helps in finding the appropriate bucket location to store our data.
     * This is very important method, as performance of HashMapCustom is very much dependent on this method's implementation.
     * @param key
     */
    private int hash(K key){
        return Math.abs(key.hashCode()) % capacity;
    }

}



/**
 * Main class- to test HashMap functionality.
 */
public class HashSetCustomEmployee {

      public static void main(String[] args) {

      HashSetCustom<Employee> hashSetCustom = new HashSetCustom<Employee>();
      hashSetCustom.add(new Employee("10", "sam"));
      hashSetCustom.add(new Employee("21", "amy"));
      hashSetCustom.add(new Employee("31", "rob"));
      hashSetCustom.add(new Employee("41", "sam"));
      hashSetCustom.add(new Employee("42", "wil"));

      System.out.println("HashSetCustom contains employee with id=21 & name='amy' : "+hashSetCustom.contains(new Employee("21", "amy")));
             System.out.println("HashSetCustom contains employee with id=51 & name='pat' : "+hashSetCustom.contains(new Employee("51", "pat")));

       System.out.print("Displaying : ");
       hashSetCustom.display();

       System.out.println("\n\nemployee with id=21 & name='amy' removed: "+hashSetCustom.remove(new Employee("21", "amy")));
       System.out.println("employee with id=51 & name='pat' removed: "+hashSetCustom.remove(new Employee("51", "pat")));

       System.out.print("Displaying : ");
       hashSetCustom.display();

   }
}

/*Output

HashSetCustom contains employee with id=21 & name='amy' : true
HashSetCustom contains employee with id=51 & name='pat' : false
Displaying : Employee[id=21, name=amy]  Employee[id=41, name=sam]
 Employee[id=42, name=wil]  Employee[id=10, name=sam]  Employee[id=31, name=rob]

employee with id=21 & name='amy' removed: true
employee with id=51 & name='pat' removed: false
Displaying : Employee[id=41, name=sam]  Employee[id=42, name=wil]
 Employee[id=10, name=sam]  Employee[id=31, name=rob]

*/
```