- Algorithm
- Apache Tomcat Server
- Collection Framework
- Collection programs
- Collections implementation
- Core Java
- Core Java Differences
- Core Java Tutorials
- Custom implementation of Data Structures
- Data Structure
- Date tutorials
- eclipse
- Exceptions
- File IO/File handling
- Garbage collection in java

Search This Blog

Search

LinkedHashSet Custom implementation in java - How LinkedHashSet works internally with diagrams and full program

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

Contents of page :

- *1) Methods used in custom LinkedHashMap in java >*
- *2) Let's find out answer of few very **important** questions before proceeding >*
- *3) Full Program/SourceCode for implementing custom LinkedHashSet in java >*

- Generics
- Hashing
- Interview questions
- iText Pdf tutorial
- Java 7
- Java 8
- Java 9
- Java QUIZ
- JavaScript
- JDBC
- JUNIT
- JVM
- Level1 programs (beginners)
- Level2 programs (intermediate)
- Level3 programs (advanced)
- Linked List
- Linux
- Matrix programs
- MongoDB
- MongoDB Java
- MsSql
- MultiThreading
- MySql
- Notepad++
- Oracle

In this post i will be explaining **LinkedHashSet** custom implementation.

*1) Methods used in **custom LinkedHashSet** in java >*

public void **add**(E value)	Add objects in **setCustom**
public boolean **contains**(E obj)	Method returns true if **setCustom** contains the object.
public boolean **remove**(E obj)	Method removes object from **setCustom**.
public void **display**()	-Method displays all objects in setCustom. **-Insertion order is guaranteed**.

Most salient feature of **LinkedHashSet** is that it **maintains insertion order** of objects. We will be internally using LinkedHashMap.

Must read: Write a program to find out substring in given string.

*2) Let's find out answer of few very **important questions** before proceeding.*

- OutOfMemoryError
- Overriding equals and hashCode
- PostgreSQL
- Producer Consumer problem/pattern
- Pyramid generation
- RegEx
- Serialization
- Thread Concurrency
- Threads
- TUTORIALS
- Windows

Q1. How LinkedHashSet implements **hashing?**

A. Method internally uses LinkedHashMap's hash method for hashing.

Q2. How **add method** works internally?

A.
```
public void add(E value){
        linkedHashMapCustom.put(value, null);
   }
```

Method internally uses LinkedHashMapCustom's put method for storing object.

Q3. How **contains method** works internally?

A.
```
public boolean contains(E obj){
        return linkedHashMapCustom.contains(obj) !=null ? true :false;
   }
```

Method internally uses LinkedHashMapCustom's contains method for storing object.

Q4. How **remove method** works internally?

A.
```
public boolean remove(E obj){
            return linkedHashMapCustom.remove(obj);
}
```

Method internally uses LinkedHashMapCustom's put remove for storing object.

REFER: LinkedHashSet Custom implementation - add, contains, remove Employee object.

3) Full Program/SourceCode for implementing custom LinkedHashSet in java >

```
package com.ankit;

/**
* @author AnkitMittal
* Copyright (c), AnkitMittal . All Contents are copyrighted and must not be reproduced in any form.
* This class provides custom implementation of LinkedHashSet(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.
* maintains insertion order.
* @param <K>
* @param <V>
*/
class LinkedHashSetCustom<E>{

   private LinkedHashMapCustom<E, Object> linkedHashMapCustom;

   public LinkedHashSetCustom(){
        linkedHashMapCustom=new LinkedHashMapCustom<>();
   }

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

   }

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

All Labels

Algorithm(34) Apache Tomcat Server(4) Arrays Java(30) Autoboxing(5) Basic java programs for beginners(15) Binary Trees(6) Collection Framework(68) Collection programs(105) Collections implementation(16) Comparable and Comparator program(22) Core Java(1032) core java Basics(38) Core java Conversions(21) Core Java Differences(11) Core Java Tutorials(12) CRUD MongoDB java(5) CRUD operations MongoDB(3) Custom implementation of Data Structures(11) Data Structure(26) Database(1) Database Tutorials(3) Date tutorials(11) DeSerialization(19) eclipse(27) Exceptions(71) File IO/File handling(71) Garbage collection in java(43) Generics(5) Hashing(8) Hibernate(1) Interview questions(14) iText Pdf tutorial(45) Java 4(1) java 5(3) Java 7(32) Java 8(80) Java 9(12)

```
  }

  /**
   * Method displays all objects in LinkedHashSetCustom.
   * insertion order is not guaranteed, for maintaining insertion order refer LinkedHashSet.
   */
  public void display(){
    linkedHashMapCustom.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 linkedHashMapCustom.remove(obj);
  }

}


/** 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 LinkedHashMap(without using java api's)- which allows us to store data in key-
value pair form.
 * It maintains insertion order, uses DoublyLinkedList for doing so.
 * If key which already exists is added again, its value is overridden but insertion order does not change,
 * BUT, if key-value pair is removed and value is again added than insertion order changes(which is quite natural behaviour).
 * @param <K>
 * @param <V>
 */
class LinkedHashMapCustom<K, V> {

   private Entry<K,V>[] table;   //Array of Entry.
   private int capacity= 4;  //Initial capacity of HashMap
   private Entry<K,V> header; //head of the doubly linked list.
```

Java keywords(11) Java Mcq(Multiple choice questions)(5) Java Programs(8) Java QUIZ(7) JavaScript(2) JDBC(74) JUNIT(7) JVM(22) Level1 programs (beginners)(34) Level2 programs (intermediate)(23) Level3 programs (advanced)(13) Linux(6) Matrix programs(10) MongoDB(87) MongoDB Java(20) MsSql(1) MultiThreading(97) MySql(7) Notepad++(6) Oracle(18) OutOfMemoryError(13) Overriding equals and hashCode(11) Pattern generating(14) PostgreSQL(1) Producer Consumer problem/pattern(10) Pyramid generation(14) Recursion(10) RegEx(7) Serialization(20) Serialization top interview questions and answers in java(18) Thread Concurrency(35) Threads(73) TUTORIALS(22) Windows(8)

```
private Entry<K,V> last; //last of the doubly linked list.

/*
 * before and after are used for maintaining insertion order.
 */
static class Entry<K, V> {
    K key;
    V value;
    Entry<K,V> next;
    Entry<K,V> before;
      Entry<K,V> after;

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


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



/**
 * Method allows you put key-value pair in LinkedHashMapCustom.
 * 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){
```

Popular Posts of JavaMadeSoEasy

- HashMap Custom implementation in java - How HashMap works internally with diagrams and full program
- Collection Quiz in Java - MCQ - Multiple choice questions
- CORE JAVA - Top 120 most interesting and important interview questions and answers in core java
- THREADS - Top 80 interview questions and answers in java for freshers and experienced(detailed explanation with programs) Set-1 > Q1- Q60
- Core Java Tutorial in detail with diagram and programs - BEST EXPLANATION EVER
- COLLECTION - Top 100 interview questions and answers in java for fresher and experienced in detail - Set-1 > Q1- Q50

```
  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);
  maintainOrderAfterInsert(newEntry);
   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;
   }
}


/**
 * below method helps us in ensuring insertion order of LinkedHashMapCustom after new key-value pair is added.
 */
private void maintainOrderAfterInsert(Entry<K, V> newEntry) {

  if(header==null){
      header=newEntry;
```

- Find sum of both diagonals in matrix - program in java
- LinkedHashMap Custom implementation in java - How LinkedHashMap works internally with diagrams and full program
- Thread/multi threading Quiz in Java - MCQ - Multiple choice questions
- EXCEPTIONS - Top 60 interview questions and answers in java for fresher and experienced - detailed explanation with diagrams Set-1 > Q1- Q25

```
        last=newEntry;
        return;
      }

      if(header.key.equals(newEntry.key)){
        deleteFirst();
        insertFirst(newEntry);
        return;
      }

      if(last.key.equals(newEntry.key)){
        deleteLast();
        insertLast(newEntry);
        return;
      }

      Entry<K, V> beforeDeleteEntry=    deleteSpecificEntry(newEntry);
      if(beforeDeleteEntry==null){
        insertLast(newEntry);
      }
      else{
        insertAfter(beforeDeleteEntry,newEntry);
      }


    }

    /**
     * below method helps us in ensuring insertion order of LinkedHashMapCustom, after deletion of key-value pair.
     */
    private void maintainOrderAfterDeletion(Entry<K, V> deleteEntry) {

      if(header.key.equals(deleteEntry.key)){
        deleteFirst();
        return;
      }

      if(last.key.equals(deleteEntry.key)){
        deleteLast();
        return;
```

JavaMadeSoeasy Archive

- ▶ 2018 (17)
- ▶ 2017 (211)
- ▶ 2016 (240)
- ▶ 2015 (722)
  - ▶ December (61)
  - ▶ November (96)
  - ▶ October (14)
  - ▶ September (13)
  - ▶ August (73)
  - ▶ July (54)
  - ▶ June (29)
  - ▶ May (53)
  - ▶ April (109)

```
     }

     deleteSpecificEntry(deleteEntry);

  }

  /**
   * returns entry after which new entry must be added.
   */
  private void insertAfter(Entry<K, V> beforeDeleteEntry, Entry<K, V> newEntry) {
     Entry<K, V> current=header;
        while(current!=beforeDeleteEntry){
              current=current.after;  //move to next node.
        }

        newEntry.after=beforeDeleteEntry.after;
        beforeDeleteEntry.after.before=newEntry;
        newEntry.before=beforeDeleteEntry;
        beforeDeleteEntry.after=newEntry;

  }


  /**
   * deletes entry from first.
   */
  void deleteFirst(){

     if(header==last){ //only one entry found.
              header=last=null;
              return;
        }
        header=header.after;
        header.before=null;

  }

  /**
   * inserts entry at first.
```

- ▶ March (93)
- ▶ February (99)
- ▶ January (28)

```
 */
void insertFirst(Entry<K, V> newEntry){

    if(header==null){ //no entry found
        header=newEntry;
        last=newEntry;
        return;
    }

    newEntry.after=header;
    header.before=newEntry;
    header=newEntry;

}

/**
 * inserts entry at last.
 */
void insertLast(Entry<K, V> newEntry){

    if(header==null){
        header=newEntry;
        last=newEntry;
        return;
    }
    last.after=newEntry;
    newEntry.before=last;
    last=newEntry;

}

/**
 * deletes entry from last.
 */
void deleteLast(){

    if(header==last){
        header=last=null;
        return;
```

```
      }

      last=last.before;
      last.after=null;
}


/**
 * deletes specific entry and returns before entry.
 */
private Entry<K, V> deleteSpecificEntry(Entry<K, V> newEntry){

      Entry<K, V> current=header;
      while(!current.key.equals(newEntry.key)){
            if(current.after==null){   //entry not found
                  return null;
            }
            current=current.after;  //move to next node.
      }

      Entry<K, V> beforeDeleteEntry=current.before;
      current.before.after=current.after;
      current.after.before=current.before;  //entry deleted
      return beforeDeleteEntry;
}


/**
 * 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)){
             maintainOrderAfterDeletion(current);
             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(){

  Entry<K, V> currentEntry=header;
  while(currentEntry!=null){
     System.out.print("{"+currentEntry.key+"="+currentEntry.value+"}" +" ");
     currentEntry=currentEntry.after;
  }

}
/**
 * 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;
}



/**
 * Method returns null if LinkedHashSetCustom does not contain 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 LinkedHashSetCustom.
     * insertion order is maintained.
     * @param key
     */
    public void displaySet(){

       Entry<K, V> currentEntry=header;
       while(currentEntry!=null){
           System.out.print(currentEntry.key+" ");
           currentEntry=currentEntry.after;
       }

    }
}




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

    public static void main(String[] args) {
       LinkedHashSetCustom<Integer> linkedHashSetCustom = new LinkedHashSetCustom<Integer>();
       linkedHashSetCustom.add(21);
       linkedHashSetCustom.add(25);
       linkedHashSetCustom.add(30);
```

```
        linkedHashSetCustom.add(33);
        linkedHashSetCustom.add(35);

          System.out.println("LinkedHashSetCustom contains 21 ="+linkedHashSetCustom.contains(21));
          System.out.println("LinkedHashSetCustom contains 51 ="+linkedHashSetCustom.contains(51));

        System.out.print("Displaying LinkedHashSetCustom: ");
        linkedHashSetCustom.display();

        System.out.println("\n\n21 removed: "+linkedHashSetCustom.remove(21));
        System.out.println("22 removed: "+linkedHashSetCustom.remove(22));

        System.out.print("Displaying LinkedHashSetCustom: ");
        linkedHashSetCustom.display();

    }
}

/*Output

LinkedHashSetCustom contains 21 =true
LinkedHashSetCustom contains 51 =false
Displaying LinkedHashSetCustom: 21 25 30 33 35

21 removed: true
22 removed: false
Displaying LinkedHashSetCustom: 25 30 33 35

*/
```

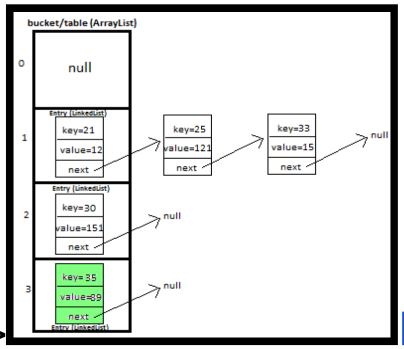
Summary of article >

In this tutorial we learned how to create and implement own/custom **LinkedHashSet** in java with full program, diagram and examples to insert and retrieve values in it.

Having any doubt? or you you liked the tutorial! Please comment in below section.

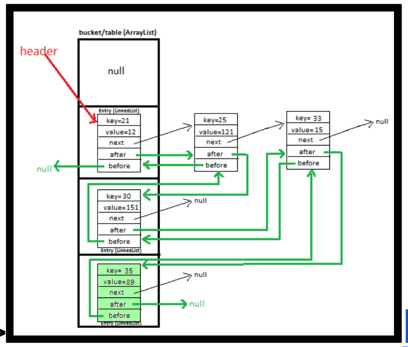
*Please express your love by liking **JavaMadeSoEasy.com** (**JMSE**) on **facebook**, following on **google+** or Twitter.*

RELATED LINKS>

1 >

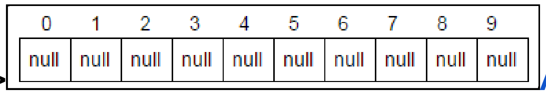
HashMap Custom implementation in java/ Develop own HashMap with full java code

>HashMap Custom implementation in java - put, get, remove Employee object

2 >

LinkedHashMap own/Custom implementation/ with full code in java

>LinkedHashMap Custom implementation - put, get, remove Employee object

3 > 

ArrayList custom implementation in java/ Create own ArrayList with full code

>ArrayList custom implementation - add, get, remove Employee object

4 >

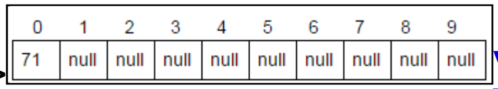
Set Custom implementation in java

>Set Custom implementation - add, contains, remove Employee object

5 >

LinkedHashSet Custom implementation in java

>LinkedHashSet Custom implementation - add, contains, remove Employee object

6 > 

Vector custom implementation in java

Labels: Algorithm Collections implementation Core Java Custom implementation of Data Structures Hashing How set works internally

Map Overriding equals and hashCode Set Single Linked List

Must read for you :

0 Comments www.javamadesoeasy.com 1 Login

Recommend Share Sort by Best

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

ALSO ON WWW.JAVAMADESOEASY.COM

JavaMadeSoEasy.com: 14 Pattern/ Pyramid generation programs in java

1 comment • 3 years ago

Pritam Bhandare — i want code for number pyramid by user input,like following1 2 3 4 5- 1 2 3 4- - 1 2 3- - - 1 2 - - - - 1

JavaMadeSoEasy.com: Eclipse - Most important and frequently used eclipse …

5 comments • 2 years ago

Ankit Mittal — Thanks Charlotte . Keep reading!

JavaMadeSoEasy.com: How garbage collection works internally in java | Mark …

3 comments • 3 years ago

Nafis Md — Hi Ankit, I am waiting your Garbage Collection full tutorial. Thanks for this wonderful unique tutorial site.

JavaMadeSoEasy.com: Collection Java - MCQ - 150 Multiple choice questions in 6 …

2 comments • 3 years ago

Ankit Mittal — We guess there must be some issue with internet connection, as they are getting loaded perfectly on our system. Thanks!

Newer Post Home Older Post