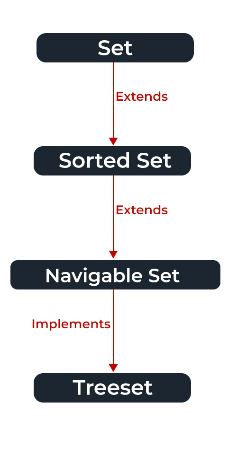
**Set-**

1. It is child interface of collection.
2. It is present in Java.util.Package.
3. If we want to represent group of individual objects as single entity where duplicates are not allowed and insertion order is not preserved then we should go for set.
4. This interface contains the methods inherited from the Collection interface and adds a feature that restricts the insertion of the duplicate elements. There are two interfaces that extend the set implementation namely SortedSet and NavigableSet.



The navigable set extends the sorted set interface. Since a set doesn’t retain the insertion order, the navigable set interface provides the implementation to navigate through the Set. The class which implements the navigable set is a TreeSet which is an implementation of a self-balancing tree. Therefore, this interface provides us with a way to navigate through this tree.

**HashSet-**

1. Underlying data structure is hashtable.
2. Duplicates are not allowed. If we are trying to insert the duplicates then we won’t get any compile.
3. Insertion order is not preserved.
4. Heterogenous objects are allowed.
5. Null insertion is possible.
6. It implements serializable and clonable interface but not random access interface.
7. Hashset is best choice if frequent operation is search operation.

Constructor-

1. HashSet hs= new HashSet();

Create the empty hashset object with default initial capacity 16 and default fill ratio is 0.75.

1. HashSet hs= new HashSet(int initialcapacity);

Create the empty hashset object with specified initial capacity and default fill ratio is 0.75.

1. HashSet hs= new HashSet(int initialcapacity, float loadfactor);

Create the empty hashset object with specified initial capacity and specified load factor.

1. HashSet hs= new HashSet(Collection c);

Load factor or fill ratio-

After loading, how much factor, new hashset object will be created that factor is called as load factor.

Example-

package com.hashset;

import java.util.HashSet;

public class HashSetDemo {

public static void main(String[] args) {

HashSet hashSet = new HashSet();

hashSet.add("ram");

hashSet.add("shyam");

hashSet.add(null);

hashSet.add(10);

hashSet.add("ram"); // return false

System.out.println(hashSet);

}

}

Output-

[null, shyam, 10, ram]

**LinkedHashSet-**

It is the child class of HashSet.

Introduced in 1.2 version.

It is exactly same as hashset but except the following difference.

|  |  |
| --- | --- |
| HashSet | LinkedHashSet |
| The underlying data structure is hashtable. | The underlyng data structure is hashtable + LinkedList (that is hybrid data structure). |
| Insertion order is not preserved. | Insertion order is preserved. |
| Introduced in 1.2 version. | Introduced in 1.4 version. |

Example-

package com.hashset;

import java.util.HashSet;

import java.util.LinkedHashSet;

public class HashSetDemo {

public static void main(String[] args) {

LinkedHashSet hashSet = new LinkedHashSet();

hashSet.add("ram");

hashSet.add("shyam");

hashSet.add(null);

hashSet.add(10);

hashSet.add("ram"); // return false

System.out.println(hashSet);

}

}

Output-

[ ram, shyam, null, 10]

Note-

1. Load factor is vary between 0.0 to 1.0 and default load factor is 0.75

2. LinkedHashSet is best choice if we want to develop cache based application where duplicates are not allowed and insertion order is preserved.

**SortedSet-**

1. It the child interface of set.
2. It we want to represent group of objects according to some sorting order and duplicates are not allowed then we should go for sortedset.

Methods-

Object first()- return the first element of sortedset.

Object last()-return the last element of sortedset.

**TreeSet-**

1. Underlying data structure is balanced tree.
2. Duplicates objects are not allowed.
3. Insertion order is not preserved.
4. All the objects will be inserted according to some sorting order.
5. Heterogenous objects are not allowed.
6. If we are trying to insert the heterogenous objects then will get run time exception saying classcastexception.
7. Null insertion is not allowed, if we are trying to insert it then will get run time error as NullPointerException.

Constructor-

1. TreeSet ts= new TreeSet();

Create the empty treeset object where elements will be inserted according to default natural sorting order.

1. TreeSet ts= new TreeSet(Comparator c);

Create empty treeset object where elements will be inserted according to customized sorting order.

1. TreeSet t= new TreeSet(SortedSet s);
2. TreeSet t= new TreeSet(Collection c);

Example-

package com.test;

import java.util.TreeSet;

public class A {

public static void main(String[] args) {

TreeSet ts = new TreeSet();

ts.add("Jay");

ts.add("ram");

ts.add("Shyam");

System.out.println(ts);

}

}

Output-

[Jay, Shyam, ram]

Example-

package com.test;

import java.util.TreeSet;

public class A {

public static void main(String[] args) {

TreeSet ts = new TreeSet();

ts.add(10);

ts.add(5);

ts.add(25);

System.out.println(ts);

}

}

Output-

[5, 10, 25]

Comparison between set Implemented classes-

|  |  |  |  |
| --- | --- | --- | --- |
| Property | HashSet | LinkedHashSet | TreeSet |
| Underlying data structure | Hashtable | Hashtable+ LinkedList | Balanced Tree |
| Insertion order | Not preserved | Preserved | Not Applicable |
| Sorting order | Not applicable | Not applicable | Applicable |
| Heterogenous objects | allowed | allowed | Not allowed |
| Duplicates objects | Not allowed | Not allowed | Not allowed |
| Null acceptance | Allowed(only once) | Allowed(only once) | we will get nullpointer exception. |

Program-1

**package** com.hashset;

**import** java.util.HashSet;

**public** **class** HashSetDemo {

**public** **static** **void** main(String[] args) {

HashSet hashSet= **new** HashSet();

hashSet.add(10);

hashSet.add(20);

hashSet.add(30);

//hashSet.add(10); //duplicates are not allowed.

//System.out.println(hashSet);

**for**(Object j:hashSet ) {

System.***out***.println(j);

}

}

}

Program- 2

**package** com.hashset;

**import** java.util.HashSet;

**public** **class** HashSetDemo1 {

**public** **static** **void** main(String[] args) {

HashSet<Integer> hashSet= **new** HashSet<Integer>();

hashSet.add(10);

hashSet.add(20);

hashSet.add(30);

**for**(**int** j:hashSet ) {

System.***out***.println(j);

}

}

}

Program-3

**package** com.hashset;

**import** java.util.HashSet;

**import** java.util.Iterator;

**public** **class** HashSetDemo2 {

**public** **static** **void** main(String[] args) {

HashSet<String> hashSet = **new** HashSet<String>(); // 16 internally size increase

hashSet.add("10"); // 16\*0.75 =12

hashSet.add("20");

hashSet.add("30"); // 13th element

hashSet.add(**null**);

Iterator<String> itr = hashSet.iterator();

**while** (itr.hasNext()) {

System.***out***.println(itr.next());

}

}

}

Program-4

package com.hashset;

import java.util.HashSet;

import java.util.Iterator;

public class HashSetDemo4 {

public static void main(String[] args) {

HashSet hashSet= new HashSet();

hashSet.add(10);

hashSet.add(20);

hashSet.add(30);

hashSet.add("velocity");

hashSet.add("pune");

Iterator itr=hashSet.iterator();

while(itr.hasNext()) {

System.out.println(itr.next());

}

}

}

Program-5

package com.hashset;

import java.util.HashSet;

import java.util.Iterator;

public class HashSetDemo5 {

public static void main(String[] args) {

HashSet hashSet= new HashSet();

hashSet.add(10);

hashSet.add(20);

hashSet.add(30);

hashSet.add("velocity");

hashSet.add("pune");

//System.out.println(hashSet.remove("velocity"));

//System.out.println(hashSet);

//System.out.println(hashSet.size());

//System.out.println(hashSet.contains("pune"));

HashSet hashSet1= new HashSet();

hashSet1.add(40);

hashSet1.add(50);

hashSet1.add(60);

hashSet.addAll(hashSet1);

System.out.println(hashSet);

}

}

Program-6

**package** com.hashset;

**import** java.util.LinkedHashSet;

**public** **class** LinkedHashSetDemo {

**public** **static** **void** main(String[] args) {

LinkedHashSet<Integer> linkedHashSet=**new** LinkedHashSet<Integer>();

linkedHashSet.add(10);

linkedHashSet.add(20);

linkedHashSet.add(30);

//System.out.println(linkedHashSet);

**for**(**int** j : linkedHashSet) {

System.***out***.println(j);

}

}

}

Program-7

**package** com.hashset;

**import** java.util.Iterator;

**import** java.util.LinkedHashSet;

**public** **class** LinkedHashSetDemo2 {

**public** **static** **void** main(String[] args) {

LinkedHashSet linkedHashSet = **new** LinkedHashSet();

linkedHashSet.add(10);

linkedHashSet.add(20);

linkedHashSet.add(30);

linkedHashSet.add("pune");

linkedHashSet.add("velocity");

linkedHashSet.add("velocity"); //not allowed

// System.out.println(linkedHashSet);

Iterator itr = linkedHashSet.iterator();

**while** (itr.hasNext()) {

System.***out***.println(itr.next());

}

}

}

Program-8

**package** com.hashset;

**import** java.util.ArrayList;

**import** java.util.HashSet;

**import** java.util.Iterator;

**import** java.util.LinkedHashSet;

**public** **class** LinkedHashSetDemo3 {

**public** **static** **void** main(String[] args) {

ArrayList arrayList= **new** ArrayList();

arrayList.add(50);

arrayList.add(75);

arrayList.add(100);

arrayList.add(75);

System.***out***.println("List is="+arrayList);

HashSet hashSet=**new** HashSet(arrayList);

//System.out.println("New Element is="+hashSet);

Iterator itr=hashSet.iterator();

**while**(itr.hasNext()) {

System.***out***.println(itr.next());

}

}

}

Program-10

**package** com.hashset;

**import** java.util.TreeSet;

**public** **class** TreeSetDemo {

**public** **static** **void** main(String[] args) {

TreeSet treeSet=**new** TreeSet(); //not applicable insertion order

treeSet.add("velocity");

treeSet.add("pune");

//treeSet.add(30); //run time will get error->

//treeSet.add(); //treeset heterogenous object is not allowed

//treeSet.add(null); //run time will get

System.***out***.println("first set="+treeSet); //sorting element

TreeSet treeSet1= **new** TreeSet();

treeSet1.add(50);

treeSet1.add(10);

treeSet1.add(100);

System.***out***.println("second set="+treeSet1);

}

}