**Collection**

A diagram of a computer program

Description automatically generated with medium confidence **HashSet**

* **HashSet used for creating collections**
* **Hashset uses a hashmap for storage.**
* Example - Whenever we add any element to HashSet, it puts element into HashMap with key as our element and value as static field PRESENT.
* Hash Set inherits the AbstractSet class and implements Set interface.

**The important points about Java HashSet class are:**

* HashSet stores the elements by using a mechanism called **hashing.**
* HashSet contains **unique elements only(No duplicates element).**
* HashSet **allows null value.**
* HashSet class is **non synchronized.**
* HashSet **doesn't maintain the insertion order.**

**Understanding HashSet:**

**public** **class** HashSet<E>

**extends** AbstractSet<E>

**implements** Set<E>, Cloneable, java.io.Serializable

{

**static** **final** **long** serialVersionUID = -5024744406713321676L;

**private** **transient** HashMap<E,Object> map;

    // Dummy value to associate with an Object in the backing Map

**private** **static** **final** Object PRESENT = **new** Object();

    /\*\*

     \* Constructs a new, empty set; the backing <tt>HashMap</tt> instance has

     \* default initial capacity (16) and load factor (0.75).

     \*/

**public** HashSet() {

        map = **new** HashMap<E,Object>();

    }

   // Other constructors and methods

}

* there is one static field PRESENT of type Object
* Whenever we add any element to HashSet, it puts element into HashMap with key as our element and value as static field PRESENT.

For Example,

import java.util.HashSet;

public class HashSetInternalDemo{

public static void main(String[] a){

HashSet<String> friendList = new HashSet<>();

friendList.add("Malay");

friendList.add("Mahesh");

friendList.add("Jay");

}

}

In above java program, I have created HashSet < String > and added 3 different Strings in it. Internally it creates HashMap < String, Object> and stores elements as shown below.

A table with text on it

Description automatically generated

Source: https://codepumpkin.com/hashset-internal-implementation/



**LinkedHashSet**

* LinkedHashSet class is a combination of Hash Table and Linked list
* Java LinkedHashSet class is **non-synchronized.**
* Java LinkedHashSet class **maintains insertion order**.

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

LinkedHashSet<String> link = **new** LinkedHashSet<String>();

link.add("john");

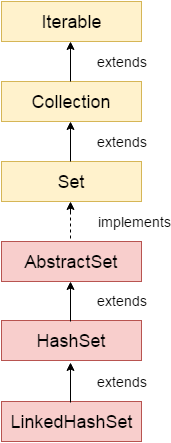
link.add("dan");

link.add("jacke");

//link.remove("dan");

System.***out***.println("Size of array is " + link.size());

System.***out***.println("Is it empty " + link.isEmpty());



**Java Tree Set**

* Collections framework provides the functionality of a **tree data structure.**
* It uses a tree data structure to store and maintain the objects.
* When we create a element in TreeSet, the JVM creates a TreeMap to store the elements in it and performs all the operations on it. It's working is similar to HashSet.

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

// **TODO** Auto-generated method stube

TreeSet<String> tree = **new** TreeSet<String>();

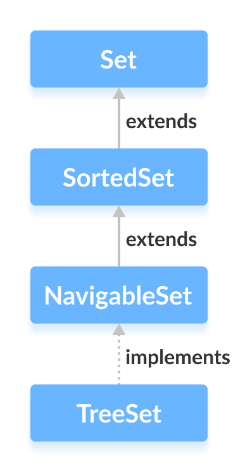
tree.add("A");

tree.add("b");

System.***out***.println(tree.size());

tree.remove("b");

}



**Hashtable**

* A Hash Table is a data structure designed to be Fast
* The reason Hash Tables are preferred instead of **arrays or linked lists** is because **searching** **adding, and deleting data** can be done really quick

**Hashtable class**

* Hashtable store **Key and value in pairs**
* It is similar to HashMap, but is **synchronized and Thread Safe.**
* Hashtable class contains only **unique elements.**
* Hashtable class **doesn't allow null value.**

Note : if your implementation is not synchronized use hashmap over Hashtable

A green screen with yellow circle and arrows

Description automatically generated

**Syntax:**

**Hashtable<K, V> num = new Hashtable<K, V>();**

**Type Parameters:**

K – the type of **keys** maintained by this map

V – the type of mapped **values**

**Example Hashmap with Integer as Key and String as value:**

Hashtable<Integer, String> star = new Hashtable<Integer, String>();

**Hastable Important Methods:**

* **Put method :** for adding element :Obj.put(1,”A”)
* **Remove methods** :for removing element
* **Put if absent methods** :if the element is not present : number.putIfAbsent(5, "E");
* **getorDefault methods :** for getting value or getting defaults values gave
* **Clear method :** To reset

**Declaration:**

public class Hashtable<K,V> extends Dictionary<K,V> implements Maps<K,V>

**Why array and linked likedlist takes time?**

* Arrays needs Index number and linked lists go Node to node both takes time for **adding, and deleting data**

**Java Hash Map:**

* HashMap store **Key and value in pairs**
* HashMap inherits AbstractMap implements the **Map interface**
* Keys should be unique. If you try to insert the duplicate key, it will replace the element into corresponding original key

**Points to remember**

* Java HashMap is non synchronized.
* Java HashMap contains only unique keys.
* Java Hashmap can store Null values
* Java HashMap maintains no insertion order.

**Hashmap Important Methods:**

* **Put method :** for adding element :Obj.put(1,”A”)
* **Remove methods** :for removing element
* **Put if absent methods** :if the element is not present : number.putIfAbsent(5, "E");
* **getorDefault methods :** for getting value or getting defaults values gave
* **Clear method :** To reset

Example :World cup years with the country which won it.

|  |  |  |
| --- | --- | --- |
| **Slno** | **Key** | **Value** |
| 1 | 1983 | India |
| 2 | 1987 | Australia |
| 3 | 2003 | Australia |
| 4 | 2011 | India |
| 5 | 2023 | India |
|  |  |  |
|  |  |  |

A diagram of a key value pair

Description automatically generated

|  |  |
| --- | --- |
| **HashMap** | **Hashtable** |
| 1) HashMap is **non synchronized**. It is not-thread safe | Hashtable is **synchronized**. It is thread-safe a |
| 2) HashMap **allows one null key and multiple null values**. | Hashtable **doesn't allow any null key or value**. |
| 3) HashMap is a **new class introduced in JDK 1.2**. | Hashtable is a **legacy class**. |
| 4) HashMap is **fast**. | Hashtable is **slow**. |
| 5) HashMap is **traversed by Iterator**. | Hashtable is **traversed by Enumerator and Iterator**. |
| 6) Iterator in HashMap is **fail-fast**. | Enumerator in Hashtable is **not fail-fast**. |
| 7) HashMap inherits **AbstractMap** class. | Hashtable inherits **Dictionary** class. |

**LinkedHashMap**

* The LinkedHashMap is just like HashMap with an additional feature of **maintaining an insertion order**
* HashMap provided the **advantage of quick insertion, search, and deletion** but it never maintained the track and order of insertion which the LinkedHashMap provides where the elements can be accessed in their insertion order.



**TreeMap class**

* TreeMap is **a red-black tree** based implementation of the Map interface
* It provides an efficient means of **storing key-value pairs in sorted order.**
* Java TreeMap maintains ascending order.

**Why treemap is needed?**

* Unlike HashMap which takes a constant time, it takes a **logarithmic time** to perform operations like searching,adding and deleting
* Memory is usage is less in treemap than hashmap which uses contaguius memory region

Source: https://tekolio.com/explaining-treemap-in-java-in-simple-english/