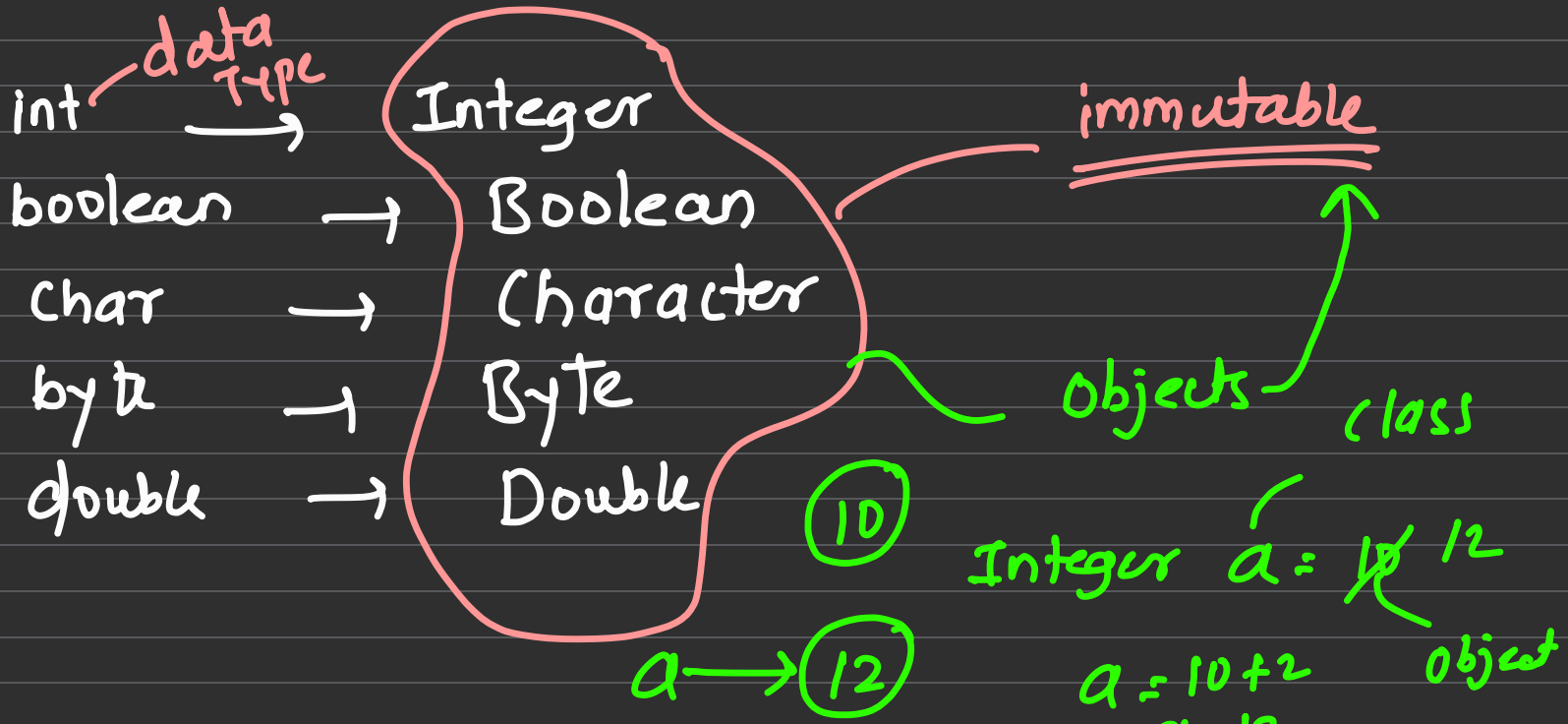
Collection Framework

"खुल जाएंगे सभी रास्ते, तू रुकावटों से लड़ तो सही,
सब होगा हासिल, तू अपनी ज़िद पर अड़ तो सही।"

Wrapper Classes

Wrapper, in general, is referred to a larger entity that encapsulates a smaller entity. Here in Java, the wrapper class is an object class that encapsulates the primitive data types.



JAVA Collection Framework

Java Collection Framework (JCF) is a set of classes and interfaces that provide ready-made data structures to store and manipulate groups of objects efficiently.

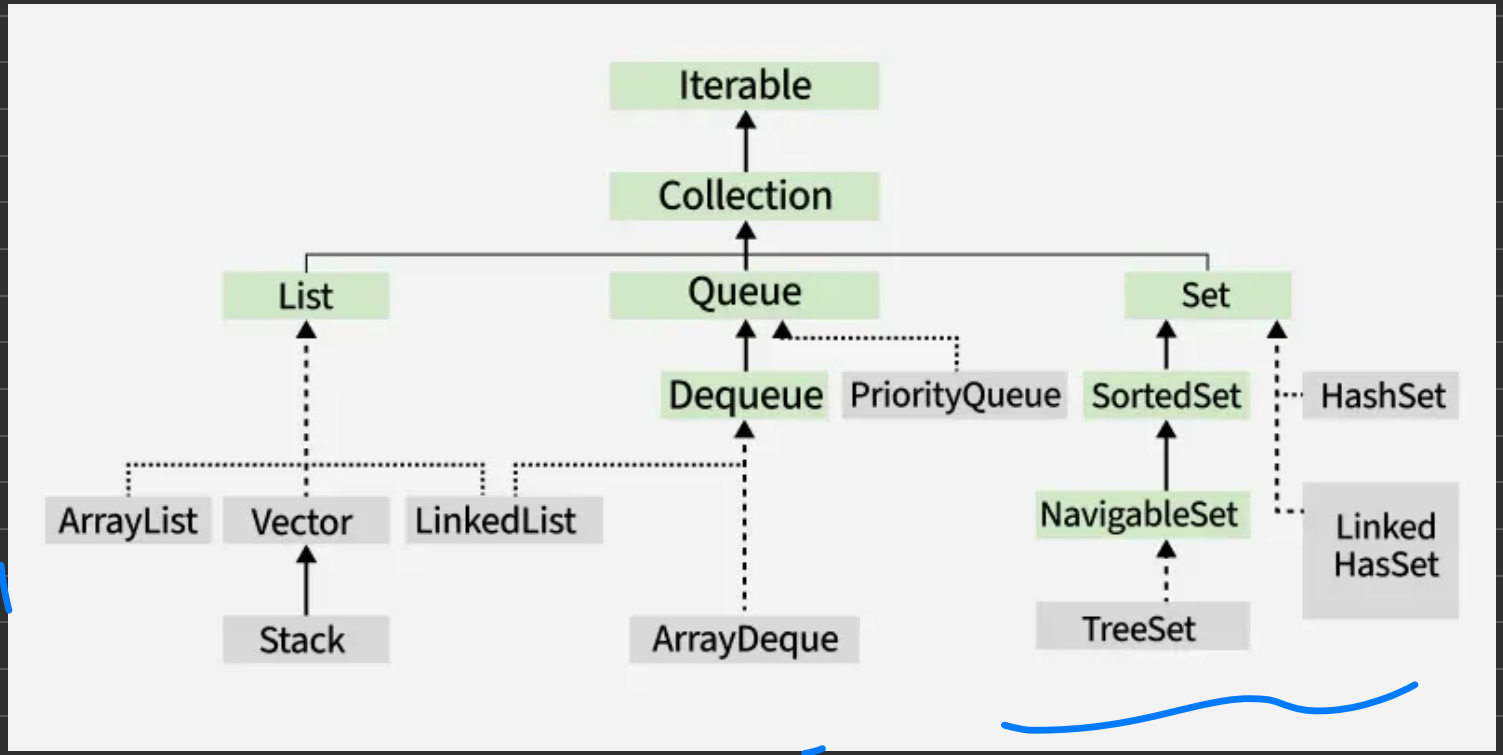
Collection

The Collection interface is the root of the Java Collections Framework,
It represents a group of individual objects as a single unit and provides basic operations for working with them. Also provides ready to use data structures

Collections

Set of utility classes which is used to implement the Collection Interface.
Also it provides some static methods to perform operation on the interfaces.

3



Java Collections Framework:

Data Structures:

List:

1. ArrayList : Dynamic size array
2. LinkedList : Doubly Linked List
3. Vector : Dynamic array + thread safe

Set:

1. HashSet : Implementation of Hashing
2. TreeSet : Self Balancing Binary Search (sorted)
3. LinkedHashSet : Hashing but insertion order is maintained

Queue

1. LinkedList : can be used as Queue (LL)
2. ArrayDeque : Array impl. of Queue
3. PriorityQueue : Heap

Deque:

1. LinkedList : dll impl of Deque
2. ArrayDeque : Array impl of Deque

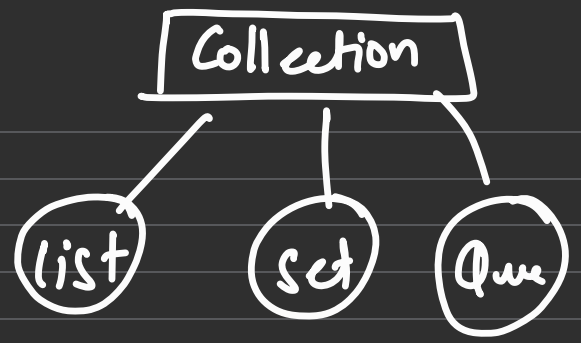
Map (key, value)

1. HashMap : impl. of Hashing
2. TreeMap : Red Black Tree
3. LinkedHashMap : insert order is stored

Collection Class:

binarySearch(), sort(), reverse(), max(), min(), fill()

Methods of Collection Interface



1. add(E e) – Adds the specified element to the collection.
2. clear() – Removes all elements from the collection.
3. contains(Object o) – Checks if the collection contains the specified element.
4. isEmpty() – Returns true if the collection has no elements.
5. remove(Object o) – Removes a single instance of the specified element, if present.
6. size() – Returns the number of elements in the collection.
7. toArray() – Returns an array containing all elements of the collection.
8. equals(Object o) – Compares the specified object with this collection for equality.

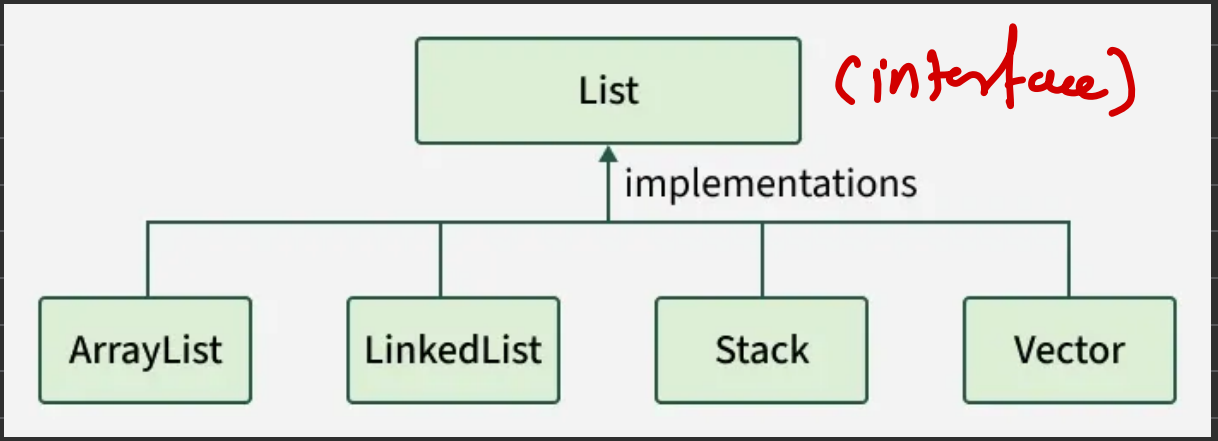
class A class B ⟶ extends

interface A class B ⟶ implements

interface A interface B ⟶ extends

List

It is used to store ordered collections where duplicates are allowed and elements can be accessed by their index.



2 3 1 6 4 4 3 2

Additional Methods of List

get(index)

set(index, value)

indexOf(Object)

lastIndexOf(Object)

ArrayList

Dynamic array where growing & shrinking of size is automatically handled by JVM

package : java.util.ArrayList

Syntax :

```
ArrayList <Integer> arr = new ArrayList<>()
List <Integer> arr = new ArrayList<>()
```

List<Integer> list = new ArrayList<>()

JVM

GC

Dynamic

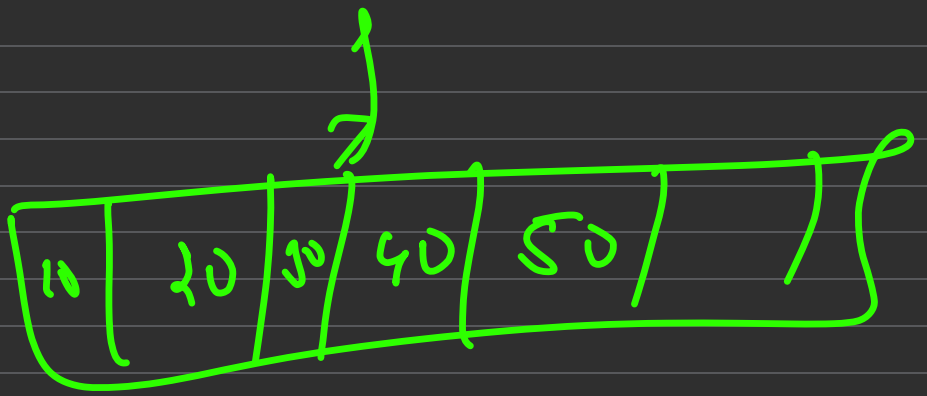
10 →

20 →

30 →

40

50



Linked List

* Collections ⟶ Doubly LL

package : java.util.LinkedList

Syntax:

wrapper

```
LinkedList <Integer> list = new LinkedList<>()

List <Integer> list = new LinkedList<>()
```

* Common

- add()
- clear()
- contains()
- size()
- remove()
- IsEmpty()
-

* List

- get (index)
- set (index, val)
- indexOf(val)
- lastIndexOf(val)

Set

$S_1 = \{1, 2, 3, 4\}$

$S_2 = \{4, 1, 3, 2\}$

$S_3 = \{1, 2, 3, 1\}$

$S_4 = \{4, 4, 2, 1, 0\}$

- order anything
- No duplicates

order
- Random
- Increase
- decrease

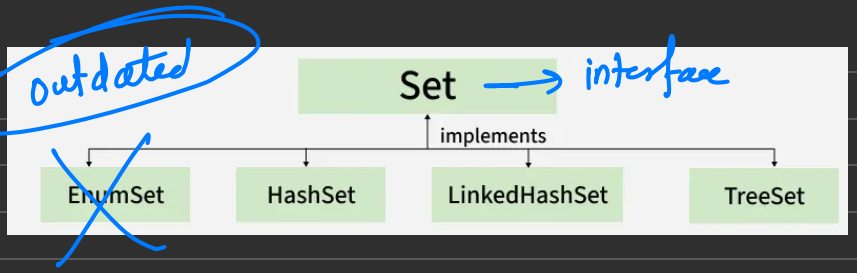
Set

It represents a collection of unique elements, meaning it does not allow duplicate values.

The set interface does not allow duplicate elements.

It can contain at most one null value except TreeSet implementation which does not allow null.

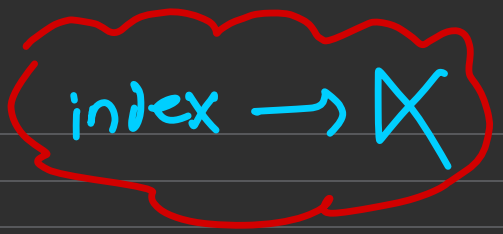
The set interface provides efficient search, insertion, and deletion operations.



* Hashset → Elements can be in any order
  insertion order is not maintained
* LinkedHashSet → insertion order is maintained
* Treeset → By default increasing / decreasing

Collection

- add()
- contains()
- size()
- clear
- isEmpty()
- remove()



↓

You cannot Traverse Using normal For loop

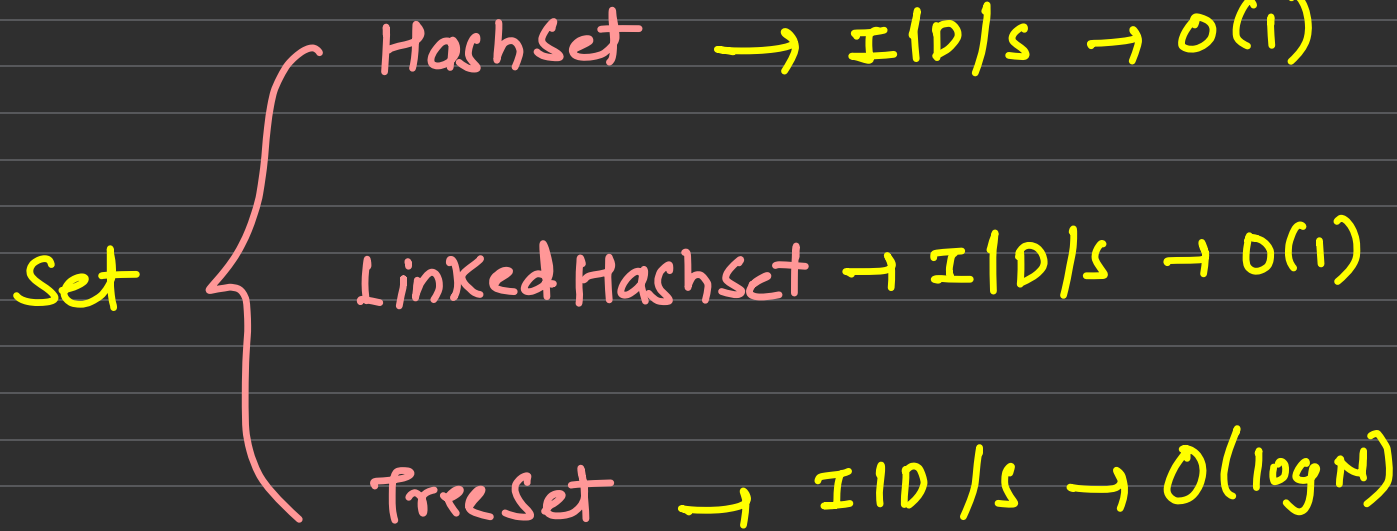
* Enhanced loop

→ for each loop

* for (i=0 ; i<n ; i++)

→ for (Integer i : s)

* Syntax :



```
Set<Integer> s = new HashSet<>()

Set<Integer> s = new LinkedHashSet()

Set<Integer> s = new TreeSet<>()
```

60

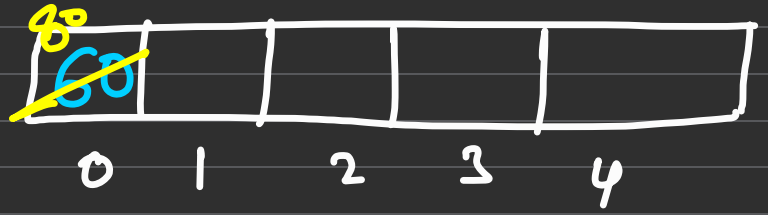
O(1)

q(0) == 60

60 == cc

N = 5

qm:



O(1)

80 % 5 = 0

60

60 % 5

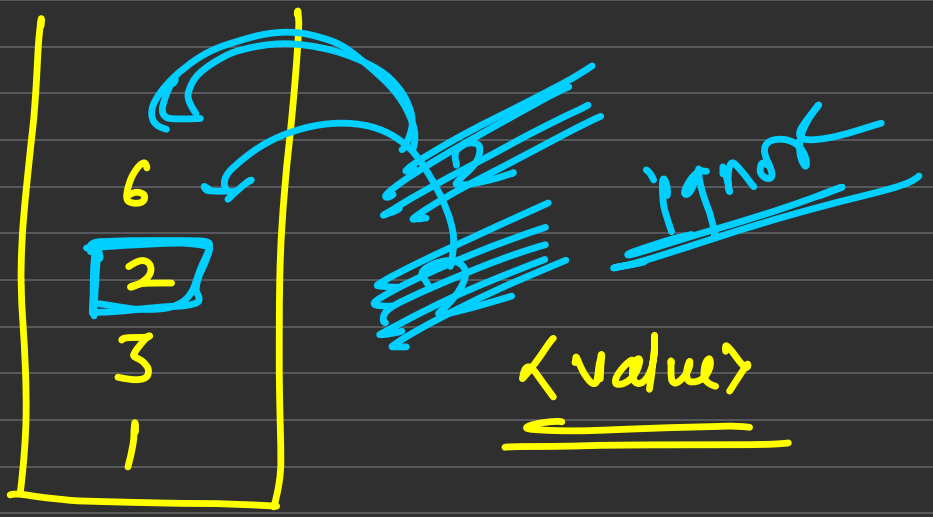
0

O(1)

h(x) = x % N

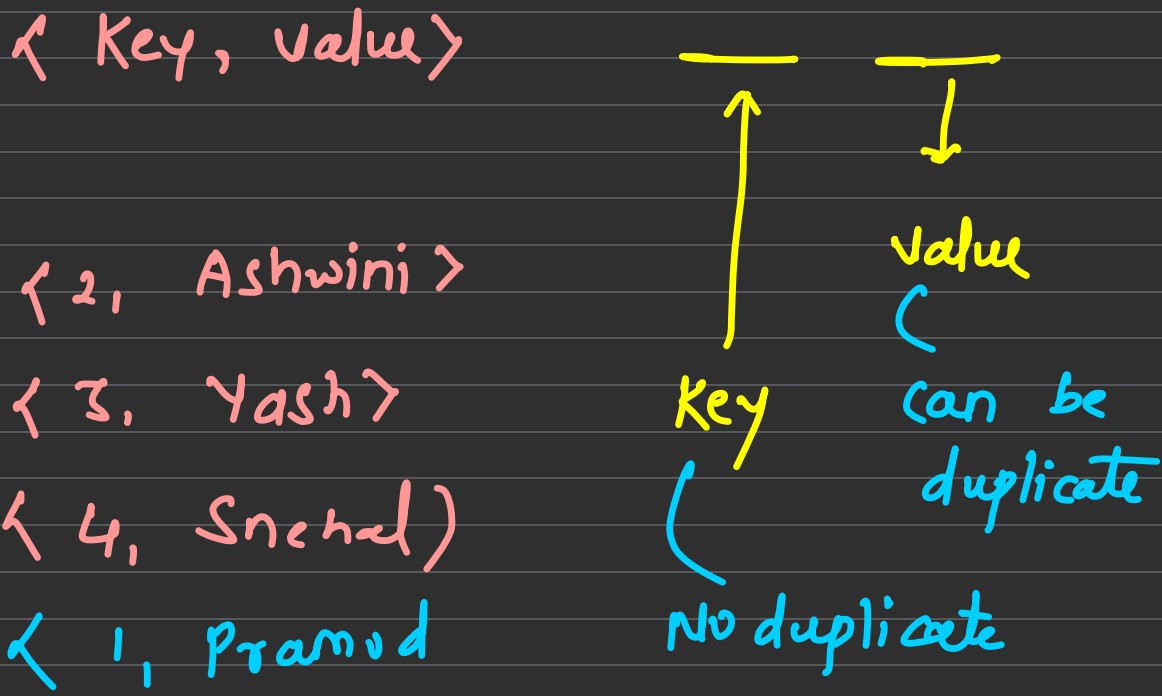
= 60 % 5

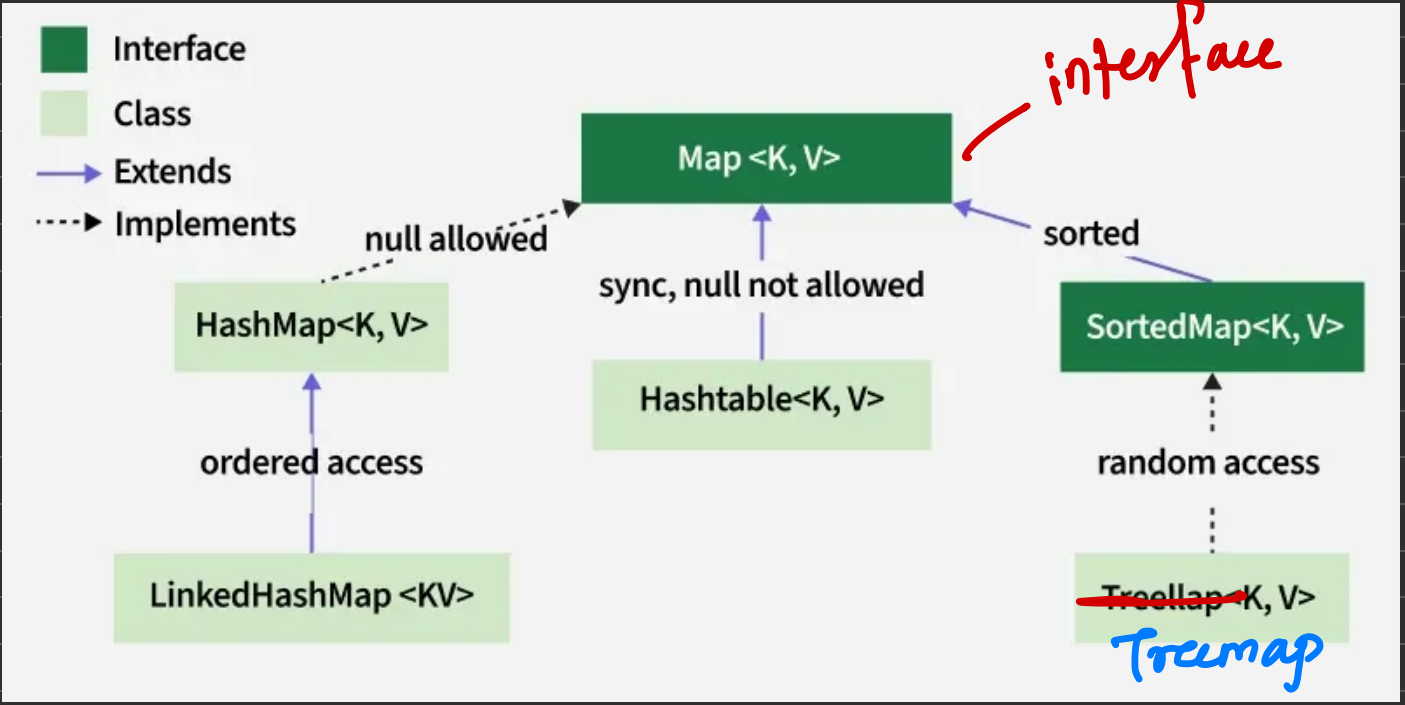
= 0



Map

represents a collection of key-value pairs, where Keys should be unique, but values can be duplicated.





map ⟶ < Key, Value > (pair)

All operations are performed on Key

insert
delete
search
update
} Key

Yash

< 9, Yash >

Key → Value

{7, Krutika}
{6, Diksha}
{4, Snehal}
{2, Yash}
{1, Pramod}

* HashMap

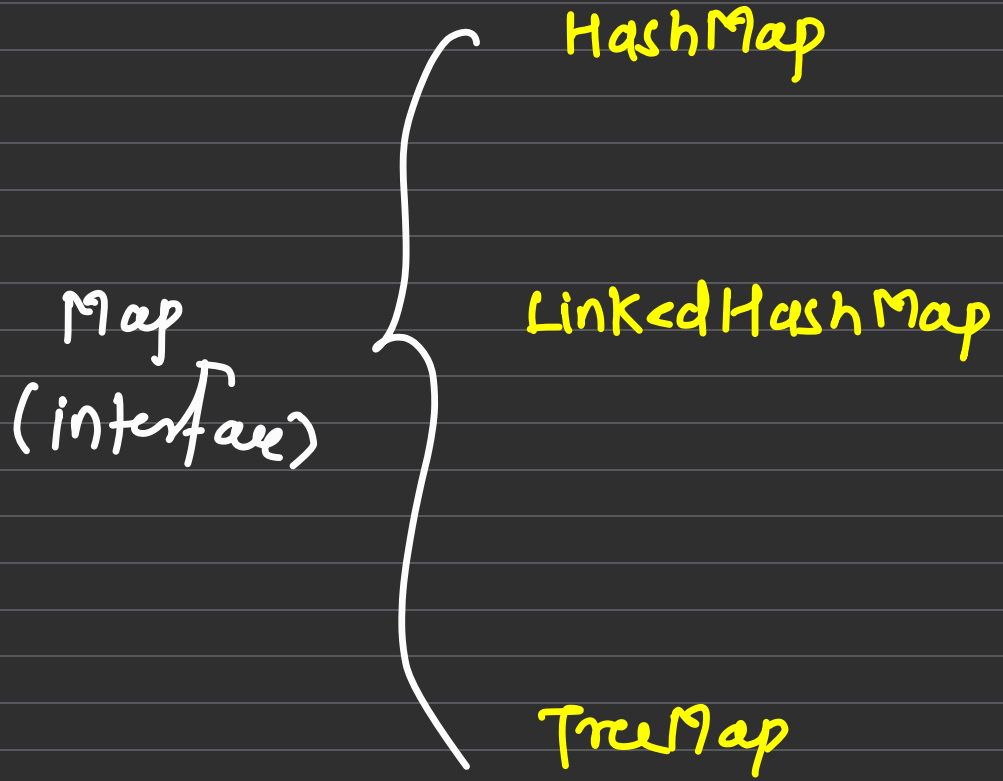
  Insertion order of key is not maintained

* LinkedHashMap

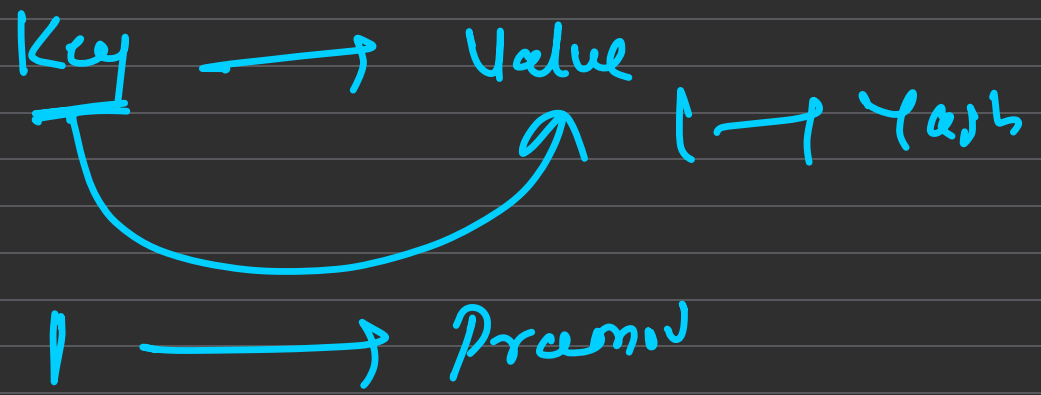
  Insertion order of key is maintained

* TreeMap

  Data is in Increasing order on the basis Key



```
Map<dataT1, dataT2> m = new HashMap<>();
Map<dataT1, dataT2> m = new LinkedHashMap<>();
Map<dataT1, dataT2> m = new TreeMap<>();
```



<7, Krutika>
<6, Dikcha>
<4, Snehal>
<2, Yash>
<1, Pramod>

* Methods of Map

* put (key, value)
  mainly adds <key, value> in map

* get (key)
  mainly gives you value mapped with given key

* isEmpty()

* size()

* containsKey (key)

- Return True if key is present else false

* remove (key)

completely Removes whole pair

* Traversing

<7, Krutika>
<6, Dikcha>
<4, Snehal>
<2, Yash>
<1, Pramod>

```
for (Map.Entry<DT1, DT2> i : M.entrySet())
{
    Key = i.getKey()
    Val = i.getValue()

}   SOP (Key)
    SOP (Val)
```

Collection Class:

sort(), reverse(), max(), min(), fill()