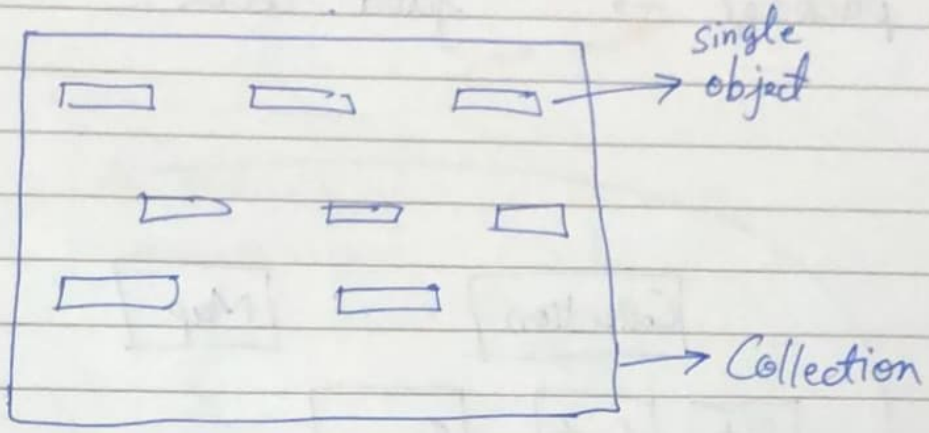
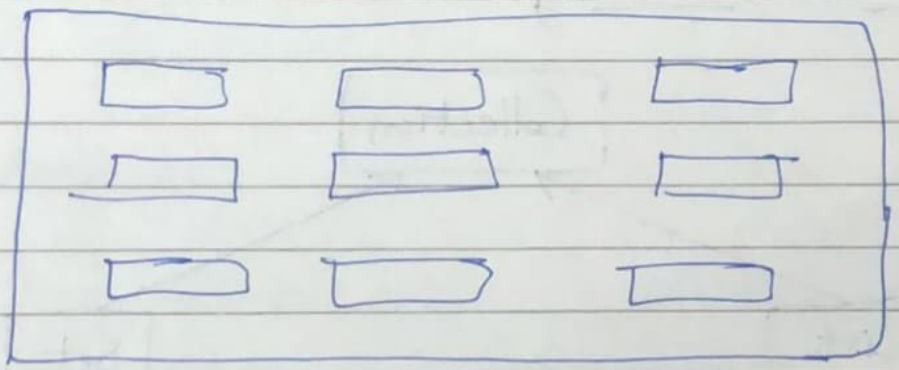
Collection Framework

→ Any group of individual objects which are represented as a single unit is known as the collection of the objects.



What is Framework ?

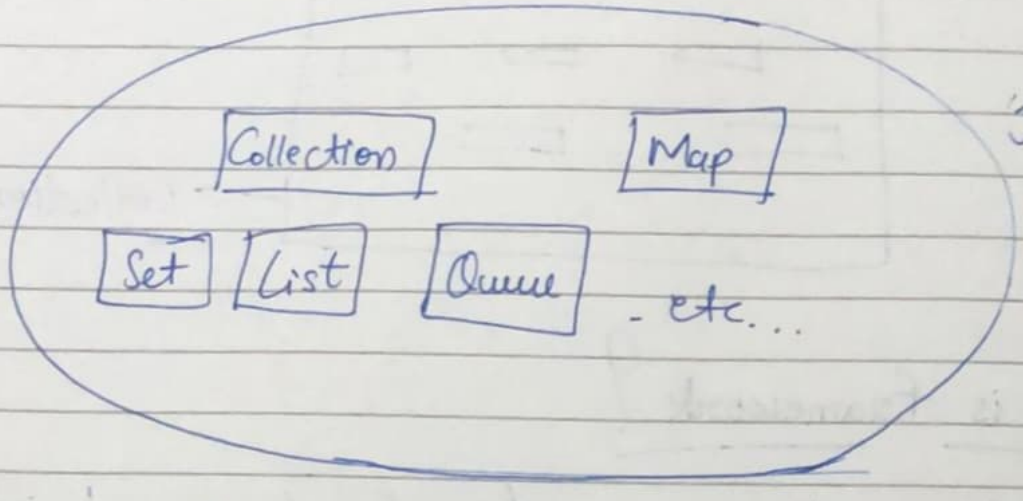
→ A framework is a set of classes and interfaces which provide a ready made architecture.



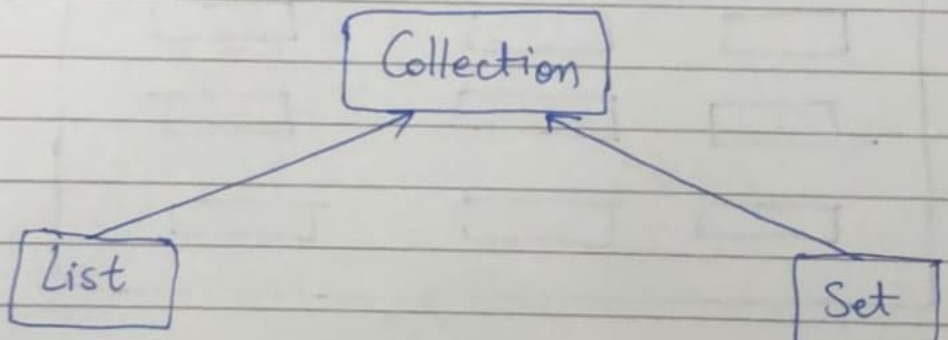
eg. Collection framework
Spring framework etc.

Collection framework is java API which provides architecture to store and manipulate group of objects.

package → java.util



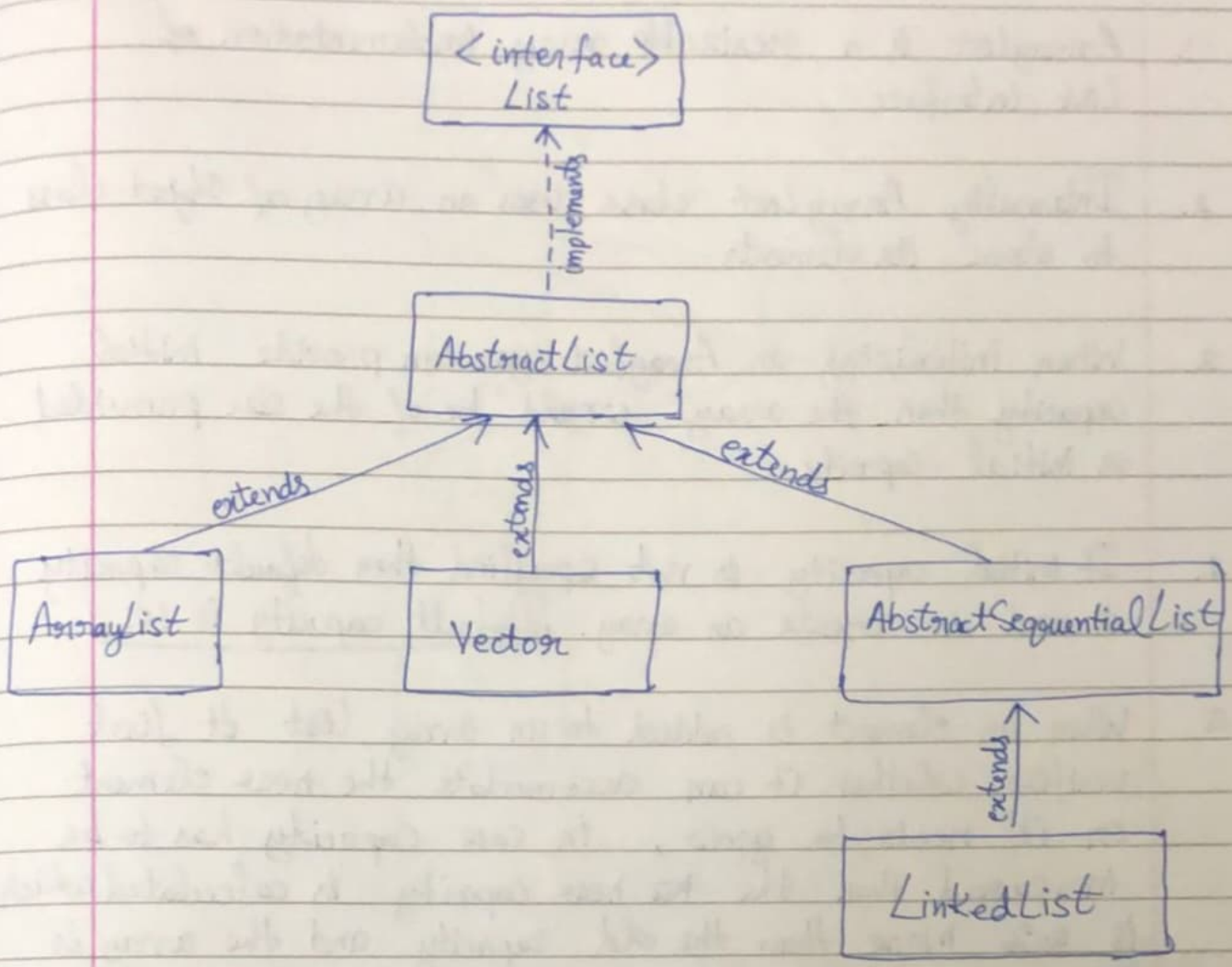
⇒ Collection Hierarchy



indexed collection
order is preserved
duplicates are allowed

non indexed collection
order not preserved
duplicates are not allowed

LIST



ArrayList	LinkedList
→ Dynamic Array implementation	→ Doubly Linked List implementati
→ adding and removing element in between is slow	→ adding and removing elements in between is fast
→ Seraching element is fast	→ Searching element is slow

Key points about internal working of ArrayList

1. ArrayList is a resizable array implementation of List interface

2. Internally ArrayList class uses an array of Object class to store its elements.

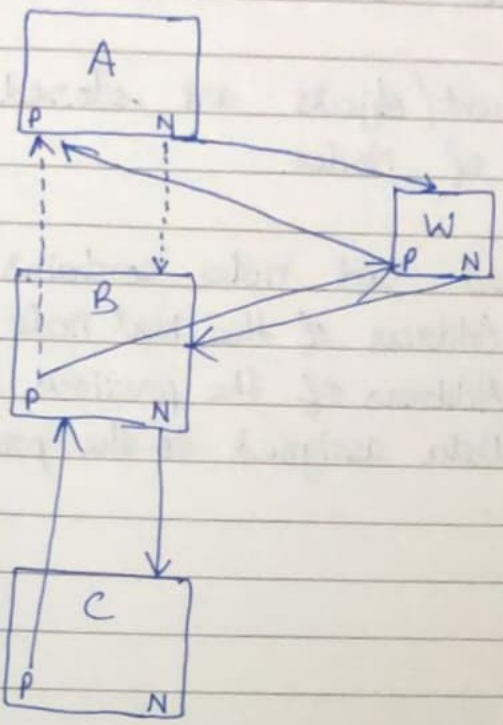
3. When initializing an ArrayList you can provide initial capacity then the array would be of the size provided as initial capacity.

4. If initial capacity is not specified then default capacity is used to create an array. Default capacity is 10.

5. When an element is added to an array list it first verifies whether it can accomodate the new element or it needs to grow, in case capacity has to be increased then the new capacity is calculated which is 50% more than the old capacity and the array is increased by that much capacity.

6. When elements are removed from an ArrayList space created by the removal of an element has to be filled in the underlying array. That is done by shifting any subsequent elements to the left.

How linked list works internally?



[A, B, C] before

[A, W, B, C] after

⇒ How to create collection:-

a) **Type Safe** → Same type of elements (objects) are added to collection

```
List<String> names = new ArrayList<>();
```

b) **Unsafe type** → Different types of elements (objects) can be added to collection

```
List list = new LinkedList();
List list = new ArrayList();
```

Key points about internal working of Linked List in Java

1. Linked list in Java is the implementation of Doubly linked list.

2. Data/element/objects are stored in linked list in the form of Nodes.

3. Doubly linked list nodes contains three fields:-
   a) Address of the next node
   b) Address of the previous node
   c) Data assigned to the present node.

Ways to create List:-

a) `List <Integer> list1 = List.of (2,4,8,10);`

b)
```
List <Integer>. list2 = new ArrayList<>();
list2. add(12);
list2. add(44);
```

c) `List<Integer> list3 = Arrays. asList (23,567, 12,99);`

Equals method in Java

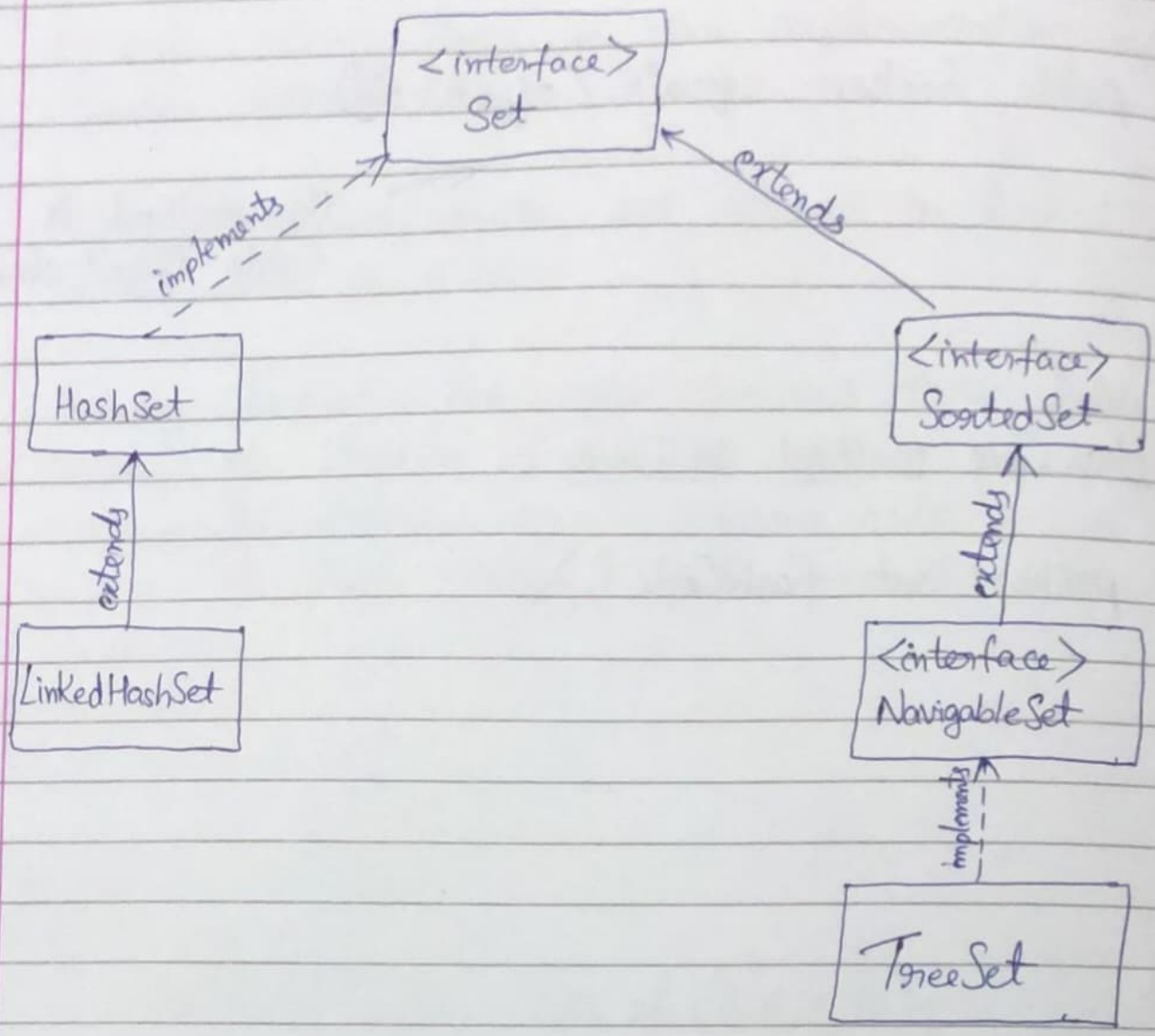
public boolean equals (Object obj)

← This method is inside Object class

Hash Code method in Java

public int hashCode ()

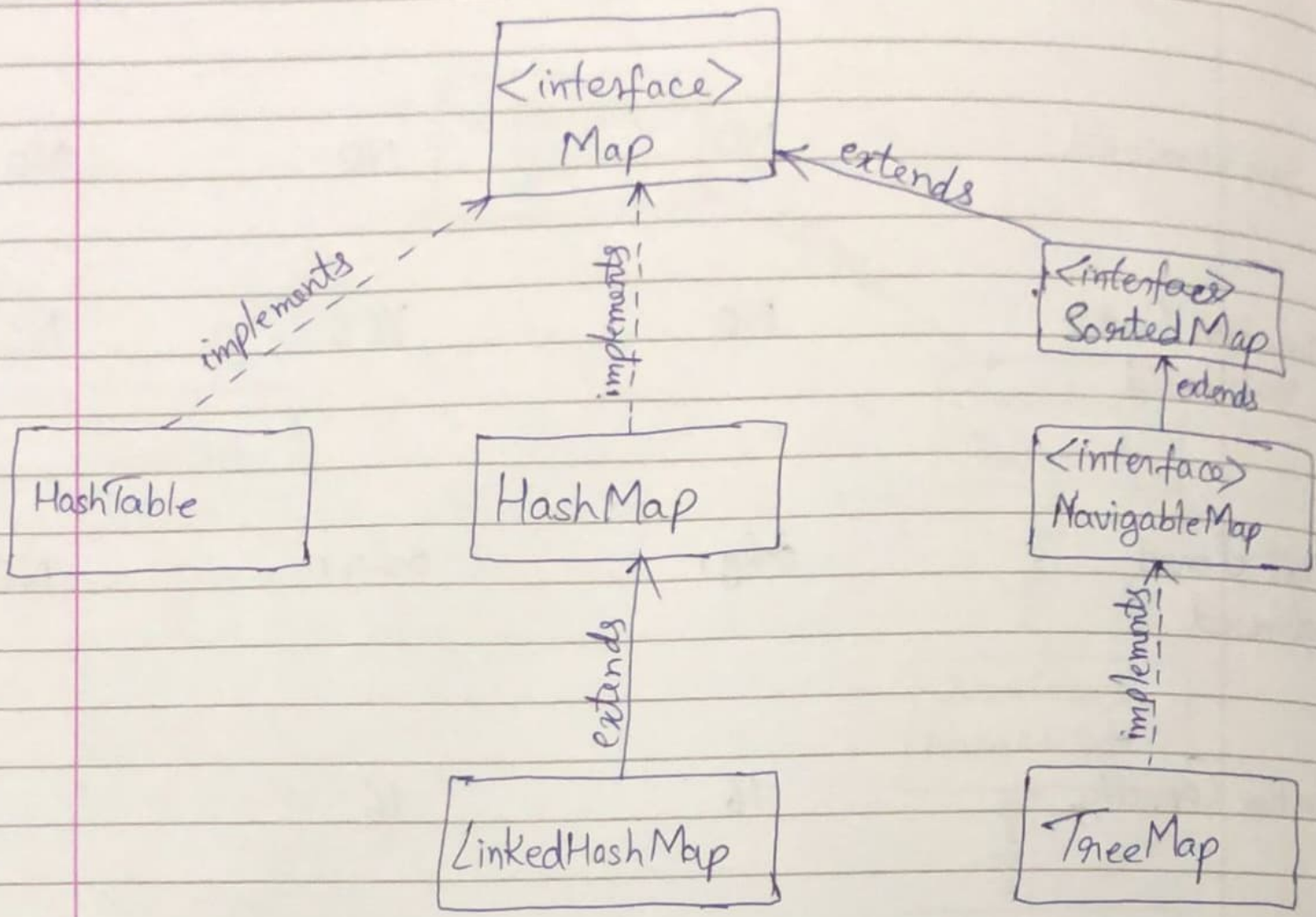
Set



- → Non indexed collection
- → Unique elements
- → Duplicates not allowed
- → Order not preserved.

	HashSet	LinkedHashSet	TreeSet
Synchronized	NO	NO	NO
Insertion Order maintained	NO	YES	NO
Null element allowed	only 1	only 1	NO
Initial Capacity	16	16	
Load Factor	0.75	0.75	
Sorted	NO	NO	YES
Internally calls	HashMap	LinkedHashMap	TreeMap

Map



→ Key value arrangement
→ Map contains unique keys.
→ Values can be duplicated.

legacy collection (Java 1.1)

	HashMap	LinkedHashMap	HashTable	TreeMap
Synchronised	NO	NO	YES	NO
Insertion Order Maintained	NO	YES	NO	
Null key allowed	one	one	NO	NO
Null Value allowed	Multiple	Multiple	0	Multiple
Default Initial Capacity	16	16	11	
Default Load factor	0.75	0.75	0.75	
Sorted	NO	NO	NO	YES on the basis of ~~Keys~~ (alphabatic/ numeric)

How HashMap Works??

```
Map<String, String> hmap = new HashMap<>();
```

Default initial Capacity = 16
Default Load factor = 0.75

Load Factor (75% or 0.75 or 3/4th)

* if HashMap reaches more than 75% of its capacity then it doubles the existing capacity.
  16 → 32

* adding data

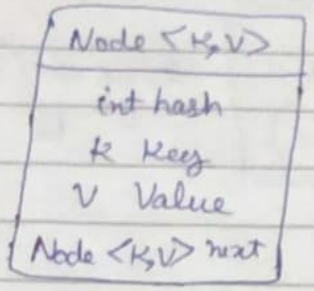
```
hmap.put("FB", "A");
```

"FB" → Key
"A" → Value

Step1 — Find hashcode of key "FB".hashCode() → 2236

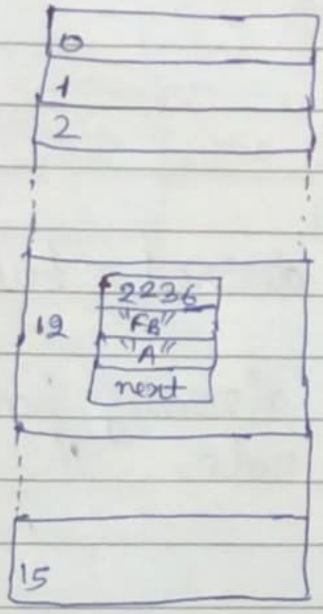
Step2 — Find the bucket index hashCode & (length-1) → 2236 & (16-1) = 12

Step3



Node created

It will add the node at the desired bucket index



* Hash Collision

hmap.put("Fa", "c");

Key

Value

Step 1

Find hashcode of Key "Fa".hashCode()

2236

Step 2

Find bucket index hashCode & (length-1)	2236 & (16-1) = 12

Step 3

Node (K,V)
int hash
K Key
V Value
Node (K,V) next

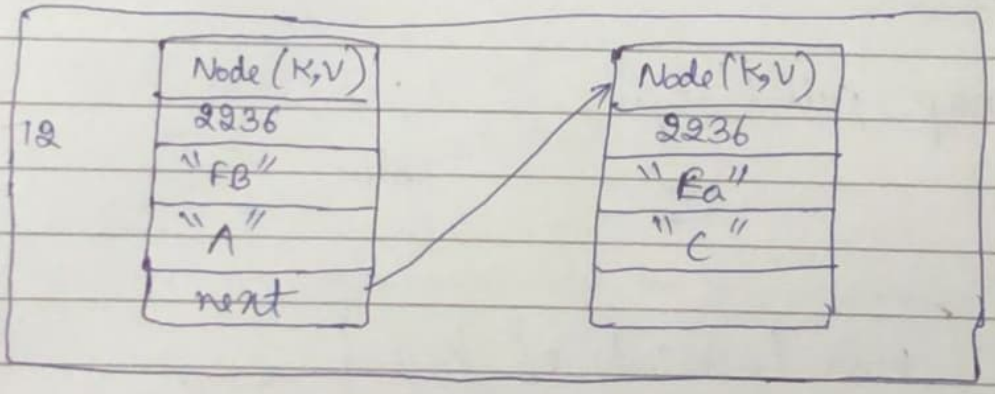
Node created

add the node at the desired bucket index.

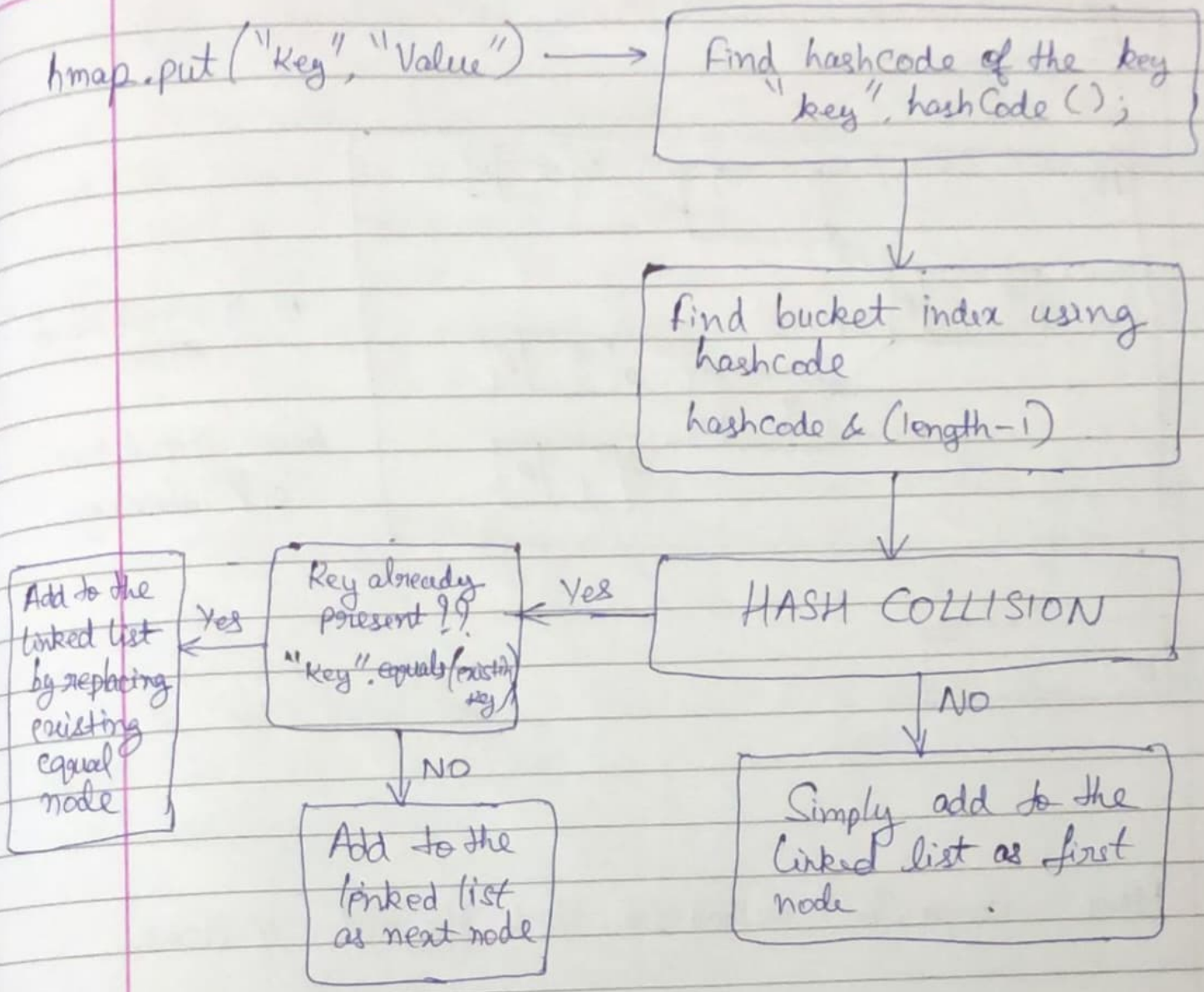
but bucket index 12 already have one node ~~at the~~ or more than one node. This situation is known as hash collision.

JVM now checks if same key is present or not.

"Ea". equals ("FB");

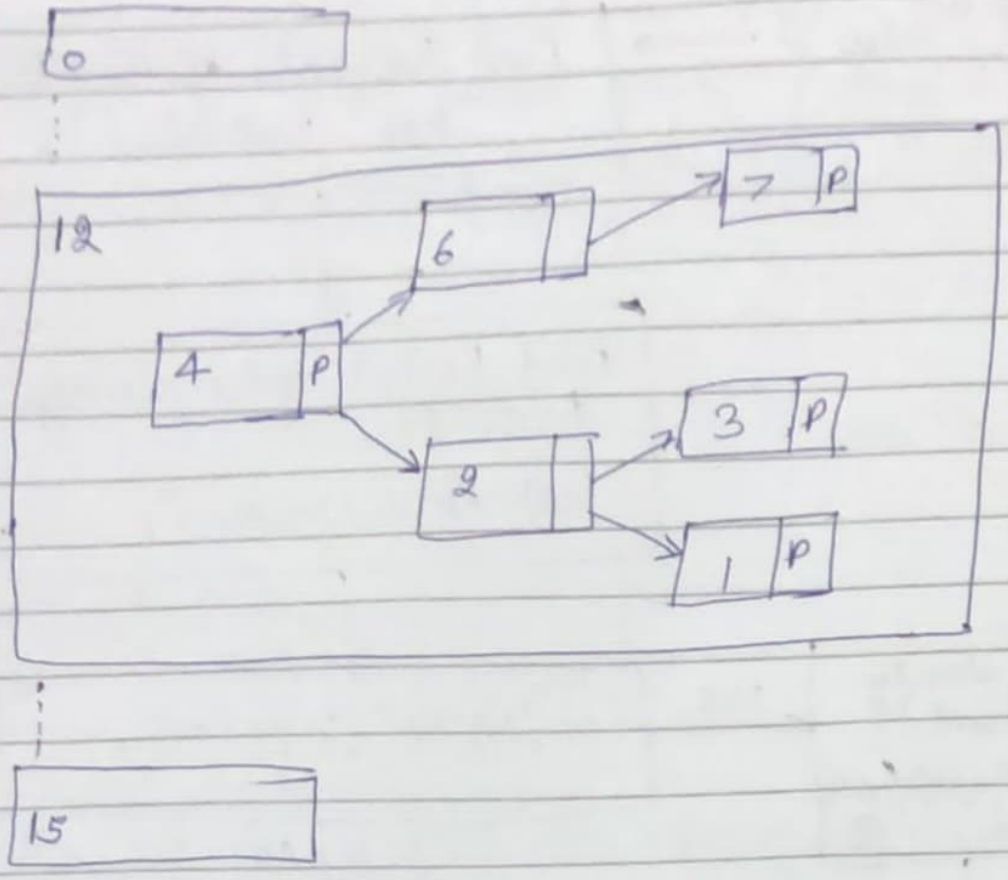


if Key is same then it replace the existing node.



* New change in Java 8

In Java 8, when we have too many unequal keys which gives same hashcode and bucket index, when the number of items in a hash bucket grows beyond certain threshold (TREEIFY_THRESHOLD = 8), content of that bucket switches from using a linked list of entry objects to a balanced tree. This improves the performance.



it is a red black tree

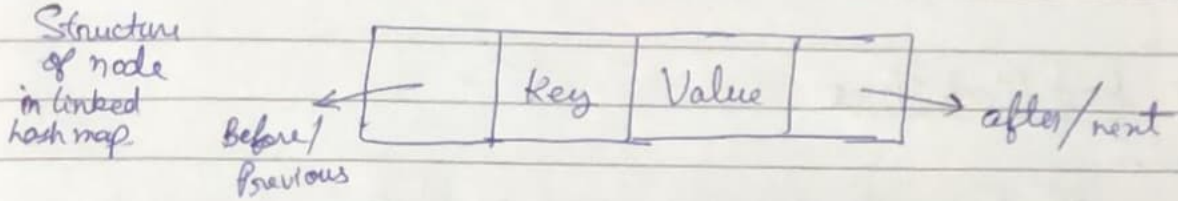
binary search tree

self balancing

Uses compareTo() method to check the order of items.

How LinkedHashMap Works ??

→ It maintains insertion order
→ It stores the value/object in a node and that node defined with a Doubley Linked List.
→ Key, Value pair
→ can have only one null key, but multiple null values allowed.

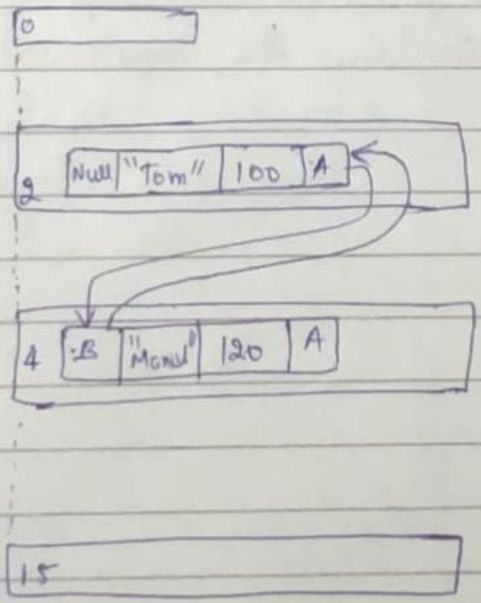



```
Map<String, Integer> map = new LinkedHashMap<>();
```

* adding data

```
map.put("Tom", 100);
map.put("Manish", 120);
```

Some like HashMap, it will calculate hashcode and bucket index



insertion order is maintained

How Tree Map Works Internally?

→ Sorted Map.
→ Does not follow hashing concept
→ Sorts the elements by keys
→ Stores in <Key-Value> pair format.
→ Red-Black Tree Concept

* Red Black Tree

a) Each node is either red or black

b) The root is black. This rule is sometimes ~~omm~~ omitted since the root can always be changed from red to black but not necessarily vice versa.

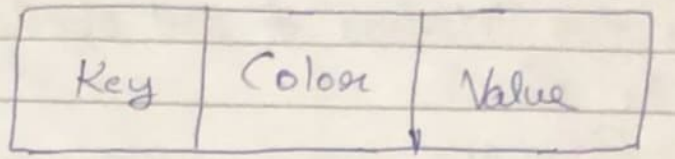
c) All leaves (NIL) are black

d) If a node is red, then both its children are black.

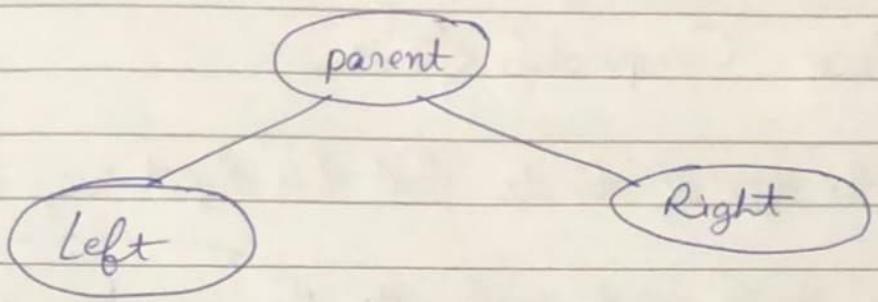
e) Every ~~path~~ path from a given node to any of its descendant NIL nodes contains the same number of black nodes.

Node in TreeMap contains three elements i.e key, value, color

Node ⇒



each node can have three references i.e. parent, left and right



1. The left element will always be logically less than the parent element.
2. The right element will always be logically greater than or equal to parent element.
3. The logical comparison of Objects is done by natural order i.e. those who implement Comparable Interface and override `CompareTo (Object obj)` method, based on the return value.

Comparable vs Comparator in Java

Java provides two interfaces to sort objects using data members of the class:-

a) Comparable
b) Comparator

⇒ Interface Comparable<T>

T → the type of objects that this object may be compared to.

The class itself must implements the java.lang.Comparable to compare its instances.

Override the method compareTo() of Comparable interface

example

String s1 = "hello";	Integer m1 = 10;
String s2 = "meklo";	Integer m2 = 30;
s1.compareTo(s2)	m1.compareTo(m2);
↳ return -5	↳ return -1
negative number	
class String implements Comparable	class Integer implements Comparable

Comparator is a functional interface.

⇒ Interface Comparator <T>

T → the type of objects that may be compared by this comparator.

Unlike comparable, Comparator is external to the element type we are comparing. Its a separate class. We create multiple separate classes (that implement Comparator) to compare by different members.

Override the method compare(T o1, T o2)

* To sort, uses Collections class

sort(List<T> list)] for comparable

sort(List<T> list, Comparator<? super T> c)] for comparator

Only of List

	Comparable	Comparator
*	Single Sorting sequence	* Multiple Sorting Sequence
*	Affects the original class	* Doesn't affect the original class
*	CompareTo() method	* Compare() method
*	java.lang	* java.util
*	Collections.sort(List)	* Collections.sort(List, comparator)

⇒ Using enhanced for loop to iterate through Collection

```
List<Employee> nameList = new ArrayList<>();

for(Employee e : nameList)
    {
        System.out.println(e);
    }
```

STREAM API

→ Introduced in Java 1.8

→ Stream API is used to perform bulk operations and process the objects of collection.

→ Stream API reduces the code length.

→ Stream is an ~~class~~ Interface In java.util.stream package.

Important Stream methods

⇒ Stream<T> filter(Predicate<? super T> predicate)

↳ returns a stream consisting of the elements of this stream that match the given predicate.

⇒ <R> Stream<R> map(Function<? super T, ? extends R> mapper)

↳ returns a stream consisting of the results of applying the given function to the elements of this stream.

⇒ collect (Collector <? super T, A, R > collector)

⇒ forEach (Consumer <? super T > action)

⇒ min (Comparator <? super T > comparator)

⇒ max (Comparator <? super T > comparator)

* We can perform operation on Collections and group of objects ie. Array with the help of Stream.

⇒ Stream<T> sorted ()

⇒ Stream<T> sorted(Comparator <? super T> comparator)

LAMBDA EXPRESSION

→ Introduced in Java 1.8

→ Lambda is anonymous function

* No name
* No Modifies
* No Return type

```
public void m1()
{
    SOP(" ");
}
```

```
() →
{
  SOP(" ");
}
```

Benefits of Lambda :-

1. Reduces the lines of code
2. Sequential and Parallel execution support by passing behaviour as an argument in methods.
3. Can only be used with Functional Interfaces.
4. To call APIs very effectively.
5. To write more readable, maintainable and concise code.

Java was always object oriented programming but after the addition of Lambda, Java started showing behaviour of Functional Programming.

```
public int sum(int a, int b)
{
    return (a+b);
}
```

```
(int a, int b) ->
{
    (a+b) ;
}
```

Important Rules of Lambda :-

1. If the body of Lambda expression contain only one statement then curly braces are optional.

2. Java compiler also infer the type of variable passed in arguments, hence type is optional.

```
public int getLength(String str)
{
  return str.length();
}
```

```
(str) -> str.length();
```

Functional Interface

1. If the interface contains only one abstract methods then it is a Functional Interface.

   eg. Runnable, Callable, Comparable etc.

2. To call Lambda we require functional interface.

3. Lambda is used to implement functional interface in very simple and short manner.