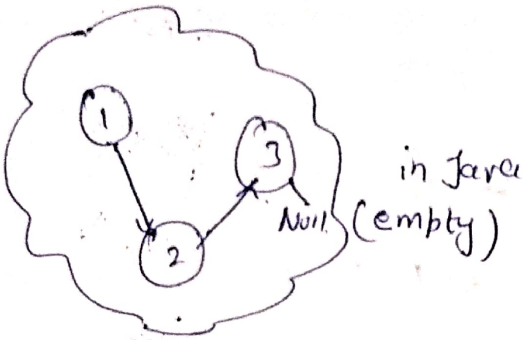
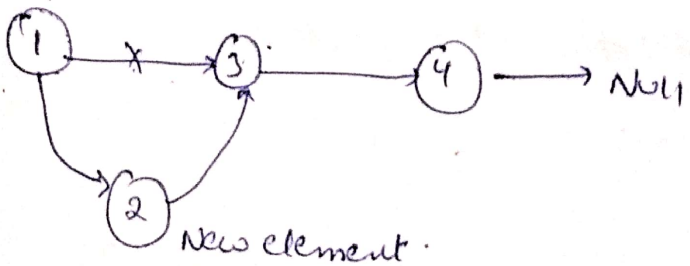
Linked List

- We have list of node that is connected with each other with Links.
- In ArrayList whenever we have an element to be ~~pre~~ insert in middle, we have to move other elements one position backward.

  ∴ Time complexity for Insert operation → O(n).
- for searching operation in Arraylist time complexity is O(1). This is because we can search element directly with index. arr[0].
- In LinkedList memory is non-contiguous; we store values at different memory chunks



Insertion in Linked List



* for Insertion operation we Prefer LinkList rather than Array List.

for Insertion we are performing just 2 & 3 operations when no. of operation are constant complexity is O(1).

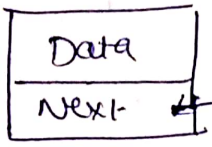
Search in Linked List. For searching we need to traverse each and every node to find the address of next node. time complexity O(n)

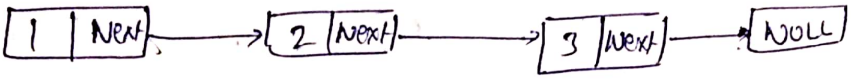
* for searching we Prefer ArrayList rather than LinkList.

Properties of Linked List

1. Variable size
2. Non-contiguous Memory
3. Insert → O(1)
4. search → O(n)

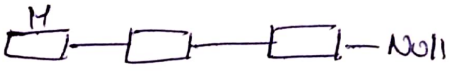
Structure of Linked List →



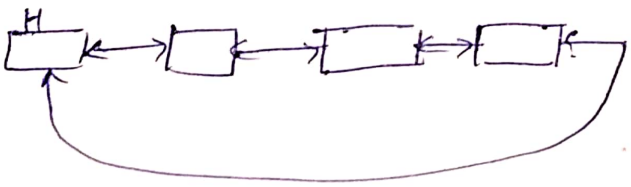


- form a chain Type structure

Types of Linked List.

1. singular 

2. Double 

3. cirular 

Linked List in collection framework.

```
import java.util.*;
  class LL {
  Public static void main(String args[])
  {
  LinkedList<String> list = new LinkedList<String>();
      list.addFirst("a");
      list.addFirst("is");
     System.out.println(list);
  }
```

o/p.
[is, a]

```
list.addLast("this");
System.out.println(list);
System.out.println(list.size()
```

o/p: [is, a, this]

(3)

```
// loop
for(int i=0; i<list.size(); i++)
{
System.out.println(list.get(i)+" -> ");
}
System.out.println("null");
```

for searching: [if(list.get(i)==value)]

o/p: [is -> a -> this -> null]

```
list.removeFist();
System.out.println(list);
list.removeLast();
System.out.println(list);
list.remove(3); Index.
System.out.println(list)
}
}
```

[a, this]

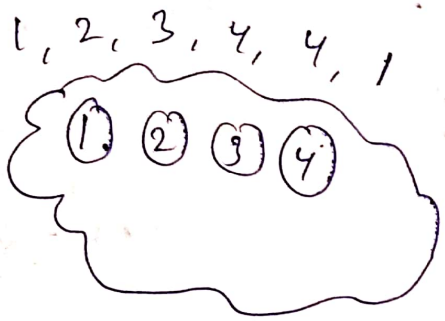
[]

→ Remove element from Particular node.

HashMap, Hash Table, Hashing.

Hash Set

- Set word is drived from Math



- Set did not allow duplicate values.
- Set is a structure which have only unique values
- To Implement this Property (Uniqueness) in Java we use Hashset
- Internal Implementation of Hashset & HashMap is done with HashTable
- HashSet is important because of its time complexity.
  - insert / Add → O(1)
  - Search / contains → O(1)
  - Delete / Remove → O(1).

  } All operations have constant Time complexity thats why it is important.

→

```
import java.util.HashSet;
Public Class Hashing.
{
Public static void main (String args[])
{
// Creating.
HashSet<Integer> set = new HashSet< >( );
```

← import java.util.Iterator

```
// Insert
set.add(1);
Set.add(2);
set.add(3);
Set.add(4);

//Search - contains function
if (set.contains(1))
{
System.out.println("set contains 1");
}
if (!set.contein(6))
{
System.out.println("does not contains");
```

→ false ; Negation ఉ. make it True & Next ది executed.

```
// Delete → Remove function
    Set.remove(1);
    if (!set.contains(1)
    {
    System.out.Println("we deleted 1");
    }
    // Size
    System.out.Println("Size of set is:" + set.size());
```

→ O/P is 2.

Because at start we add 4 element but it will stay only 3, Because 1 is duplicate & we remove 1 also.

```
// Print all elements
    System.out.println(set);
```

→ O/P [1, 2, 3]

//Iterator → used to traverse set ; Beeaz set does not have any index; It uses a special Iterator. For that import Iterator.

import java.util.Iterator

iterator is a method of set.

* Iterator it = set.iterator();

//Iterator has two function ① hasNext ; next

Exp: set = [1, 2, 3]

it.next() → 1
it.next() → 2
it.next() → 3

initially it = NULL.
Next function will point to 1st element in set, then it element point to 2..

• set never store values in Particular order
if values 1, 2, 3
out values 2, 3, 1

hasNext return true & false value.

it.hasNext(); → if there is an element in set it will return True otherwise false.

start → It will give true
Because it hold True value

Loop while (it.hasNext()) {
Print (it.next());
}. //

* while (it.hasNext()) {
Print (it.Next());
}.

HashMap

- HashMap is a structure that store 2 type of information (1) key (2) value
- The senerio where we need to store 2 pair of values we use HashMap

Exp :- we want to save Roll no. of student

Roll No. (key)	Names (values)
64	Neha
65	Kritika
66	Sawan
78	Rohan
71	Sohail

The column which have unique value serve as key. Name serve as values. Value can be duplicate But key is always unique

Exp2 Petrol Pumps

Fuel (key)	Price (value)
CNG	70
Diesel	80
Petrol	90.

- Fuel type is unique Therefore take it as key
- Price could be same ∴ Price serve as value.

```
import java.util.HashMap;
Public class Hashing
{
Public static void main(string arg[])
{
// country(key), Population (Value)
HashMap <String, Integer> map = new HashMap<>();
```

(String ↑ key Type, Integer ↓ value type)

// Insertion

```
map.put("India", 120);
map.put("US", 30);
map.put("China", 150);
System.out.println(map);
```

o/p
{China=150, US=30
India=120}

(to insert key)
```
map.put("China", 180);
System.out.println(map);
```

- Put function is used for insertion.
- [Hashmap are unordered Pair that why ordering is changed in Hashmap]

// In this case value of Population of China will be updated.

map.Put()

- key already exist → Update value
- key doesn't exit → New Pair Inserted.

// Search operation → containskey (key value) function (lookup)

```
if map.containskey("china")
{
System.out.Println("key is Present in the map");
}
else
{
System.out.Println("key is not Present in the map");
}
```

// After searching the key if we want to know about value of key we use the function get

- get()
  - key Exist → Print value of key
  - key doesn't exit → Null.
- containskey
  - Key Exist → True
  - does not exit → False.

System.out.println(map.get("China")); // key exist
System.out.println(map.get("Indonesia"); // key does not exist

o/p → 180
NULL

// Iteration in HashMap. (Traversing)

For loop syntax for HashMap.

```
for(Map.Entry<Integer, Integer> e : Map.entrySet())
```

- Integer (String, Integer) → Type of key
- Integer → Type of value
- e → element (variable)
- : (colon)
- entrySet() → give set version of map.
  - each pair is stored in variable e

//

```
for(Map.Entry<String, Integer> e : map.entrySet())
{
System.out.println(e.getKey());
System.out.println(e.getValue());
}
```

China
150
India
120
US
30

→ O/P

// another Method.

```
Set<String> keys = map.keySet();
for(String key : keys)
{
System.out.println(key + " " + map.get(key));
}
```

- keySet.
- key → key value
- map.get(key) → return the value corresponding to the key in map like for India key corresponding value is 120.

// Remove pair in map

```
map.remove("China");
System.out.println(map);
```

→ Remove both key & value.

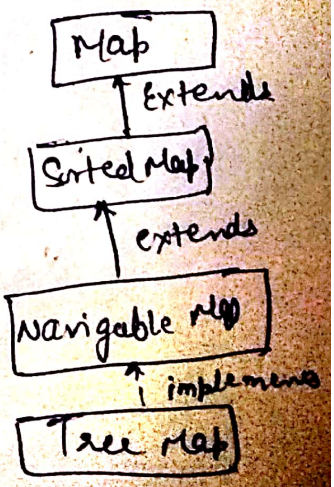
Tree Map

Treeset
Java Map Interface

TreeMap

① Java TreeMap class is a tree Based Implementation.
② Provide mean of sorting key-value Pair
Red- Black → Algorithm (self Balancing Binary search Tree
- color of every node in the Tree is either Red or Black.
- Root Node must be Black in color
- Red node cannot have Red color neighbour node
- All the Paths from root node to null should consist the same no. of Black nodes

③ Java TreeMap Contains only unique elements
④ Java TreeMap maintains ascending order..
⑤ TreeMap is a class of collection framework



import java.util.TreeMap ← Package / class.

(6) // Creation

Syntax TreeMap<key, value> numbers = new TreeMap<>();

Exp TreeMap<String, Integer> evenNumbers = new TreeMap<>();
en

(7) // Insertion → Put() → Insert specific key/value
PutAll() → Insert all entries from Map
PutIfAbsent() → Insert if specified key is not present in map.

Exp
```
en.put("Two", 2);
en.put("Four", 4);
en.putIfAbsent("Six", 6);
System.out.println("TreeMap of even no." + en);
```
→ o/p { Two=2, four=4, six=6 }

(Variable Store all values) → numbers.putAll(en)
System.out.println("TreeMap of numbers" + numbers);

(8) // Access TreeMap elements → ① entrySet(); → Return set of all keys & values
② keySet() → Return set of all keys
③ values() → Return set of all maps of Tre

```
System.out.println("Key/value Mapping:" en.entrySet());
System.out.println("keys:" + en.keySet());
System.out.println("values": en.values());
```

o/p:- Key/value Mapping = [Two=2, four=4, Six=6]
keys = [Two, Four, Six]
values = [2, 4, 6]

(9) get() → (1) value associated with key
(2) Return Null if key not present.

getOrDefault() – Return value associated with key
– Return specified value as default

```
int value1 = en.get("Two")
System.out.println("using get():" + value1);
int value2 = en.getOrDefault("five", 5);
System.out.println("using getOrDefault():" + value2.
```

[o/p using get():3]
[o/p using getOrDefault():5]

(10) Remove TreeMap Elements – remove(key) ← Remove value associated with key
– remove(key, value).

```
int value = en.remove("Two");
```

Print → value

```
boolean result = en.remov("six", 6);
```

Print → Result → True / false.

(11) Methods for Navigation.

firstKey() → Return the 1st key of Map
firstEntry() → Return the key/value mapping of the first key of map
lastKey() → Return last key of Map
lastEntry() → Return the key/value mapping of Last key.

⟷

TreeSet

- It implement SET Interface
- use Tree for storage
- Contain only unique elements
- does not allow null element
- Treeset maintain ascending order
- Implemented using Binary search Tree.

Syntax :- (1) creating

```
import java.util.TreeSet
TreeSet<Integer> numbers = new TreeSet<>();
```

Methods

(2) Insert Elements

add() → insert specified elements

addAll() → Insert all elements of specified collection

```
numbers.add(2);
numbers.add(4);
numbers.add(6);
System.out.println("Treeset:" + numbers);   o/p [2,4,6]
TreeSet<Integer> num = new TreeSet<>();
num.add(1);
num.addAll(numbers);
System.out.println("New set:" + num);   o/p [1,2,4,6]
```

(3) Access Treeset elements → Iterator() ⊲ import java.util.iterator. → hasNext() → next()

```
Iterator<Integer> iterator = numbers.iterator();
System.out.println("Treeset using iterator");
while(iterator.hasNext()){
    System.out.println(iterator.next());
}
```

④ Remove elements → remove()
removeall()

boolean val1 = numbers.remove(6);
boolean val2 = numbers.removeAll(numbers);
Print → val1 & val2.

⑤ Navigation → first() → Return 1st element
last() → Return Last element

int F1 = numbers.first();
int last = numbers.last();
Print → F1 & Last.

⟷

Java Map Interface:

- Map contains key/value Pair.
- Map is useful if you have to search, update or delete elements on the Basis of a key.
- There are Two Interface :- Map & Sorted Map
- classes → HashMap & TreeMap

⟷