Collection → Group of individual objects which represent a single unit.

Framework → set of classes & interface which provide a readymade architecture.

Ex. in Java Collection Framework → HashMap, ArrayList, HashSet, Linked List, etc.

Need for a separate collection framework (Java)

Can we create separate class for each DS & use?

Yes but each class may be implemented differently.

Eg → ArrayList $\longrightarrow$ al.add(2)
HashSet $\longrightarrow$ hs.insert(5)
Stack $\longrightarrow$ st.in(6)

insertion

If multiple classes implement same interface then functionalities are easy to remember.

Advantages of Collection Interface

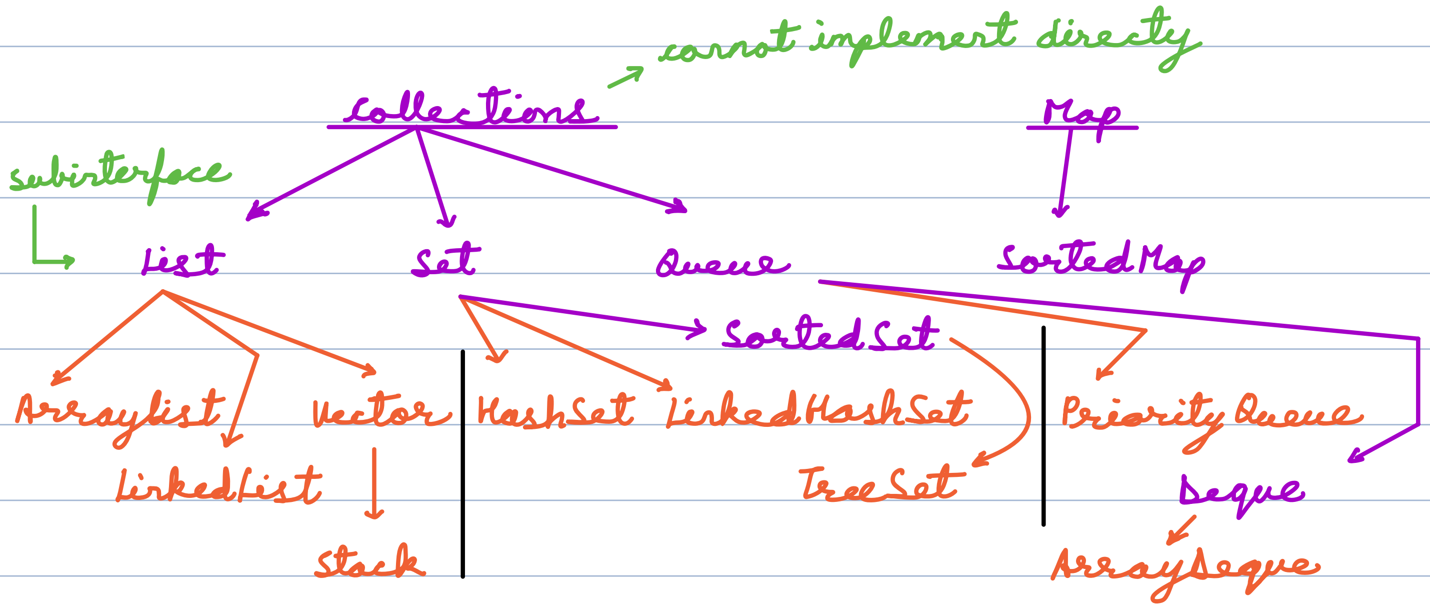
1) All the classes that implement the interface will have same set of methods.

2) Implements abstraction & hence save programmer's effort.

Hierarchy of Java Collection Framework

java.util package contains all classes & interface required by collections framework.

Interface → Blueprint of class.
Methods are only declared.
Objects cannot be created.



Iterable Interface → 1) Root interface for entire collection framework.

2) Main functionality → It provides iterator for the collections.

Methods of Collection Interface

1) add() 2) size() 3) remove()
4) iterator() 5) addAll() 6) removeAll() 7) clear()

List interface → 1) Child of collection interface.

2) Store ordered collection of object.

3) Allow duplicate data to be present.

public interface List<E> extends Collection<E>;

1) **ArrayList** → 1) Dynamic array i.e. resizable.

size = 10
load factor = 0.7 } $10 * 0.7 = 7$

⇒ If the arraylist reaches >= 7 elements it doubles the size.

size = 20
lf = 0.7 } $20 * 0.7 = \underline{14}$

2) Indexed based access available. → TC = $\underline{O(1)}$

al.get(2)
↑
index

---

2) **Vector** → 1) Provides dynamic array but it is slower compared to arraylist.

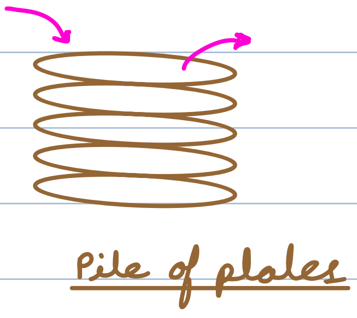
2) Identical to arraylist in terms of implementation.

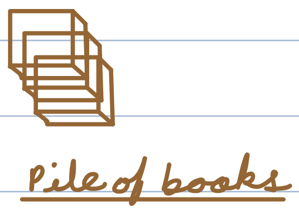
Arraylist → Non-synchronized

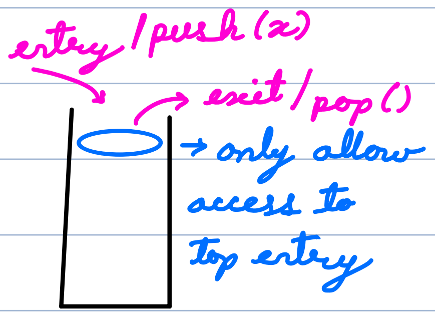
Vector → Synchronized (one task at a time)

---

3) Stack → 1) It extends vector class.
2) It is based on the basic principle LIFO (last in first out).

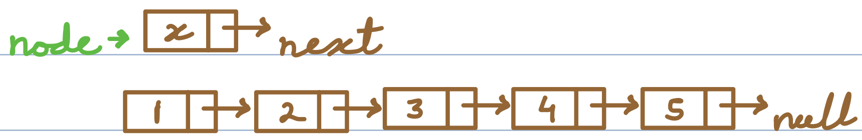






---

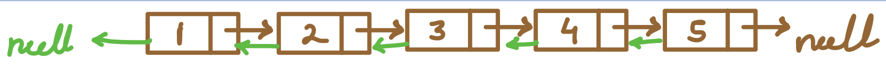
4) Linked List → Implements linked list DS which is linear DS where elements are stored in non-contiguous memory allocation.

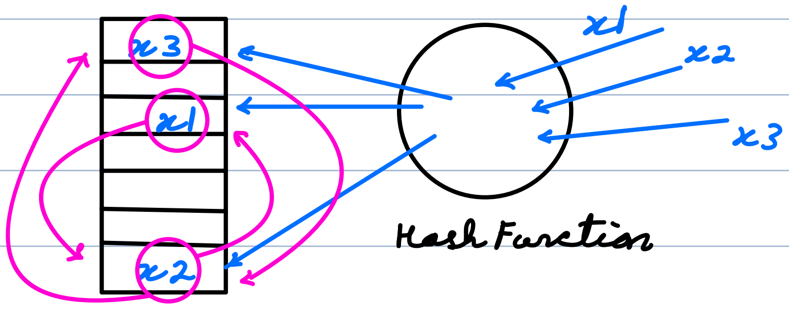


Set Interface → 1) Child of collection interface.
2) It store unordered collection of objects & do not allow duplicate elements.
3) We can store atmost 1 null value.

1) HashSet → 1) Widely used class for set interface.
2) Store data using hash fuction.

2) Linked HashSet → 1) Ordered version of HashSet
2) It maintains a doubly linked list across all elements.





3) <u>TreeSet</u> → 1) It is a implementation of sorted set interface.
2) Simar to set but maintains sorted order of data.
3) Use Tree data structure for storage.

<u>Map Interface</u> → 1) It is part of java util package but not subpart of collection interface.
2) Maintains mapping between a <u>key</u> & a value.
unique

1) <u>HashMap</u> → 1) Widely used class for map interface.
2) Store data using hash fuction.
3) Unordered

2) <u>LinkedHashMap</u> → 1) Ordered version of HashMap
2) It maintains a doubly linked list across all elements.

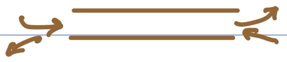
3) <u>TreeMap</u> → 1) It is a implementation of sorted map interface.
2) Simar to map but maintains sorted order of keys.

3) Use Tree data structure for storage.

Queue Interface → 1) Sub interface of collection interface.
2) Usually store data on principle of FIFO (first in first out)

1) Priority Queue → 1) Maintain order wrt priority
(aka Heaps) 2) Removal happens wrt priority,
also know as — eg largest, smallest, etc.

2) ArrayDeque → 1) It implements Deque Interface (child of Queue Interface).
2) Provides entry & exit from both sides.



Comparable

Arraylist → al = {2 5 10 3 8}
Collections.sort(al) → {2 3 5 8 10}
sort wrt natural ordering of data

```
class Person {
    String name;
    int age;
    Person(string n, int a) {
        name = n
        age = a
    }
}

    ArrayList<Person> al = new ArrayList<>();
              :
        Collections.sort(al) → Error!
```

Comparable → Defines the natural ordering for a class

```
class Person implements Comparable<Person>{
    String name;
    int age;
    Person(String n, int a){
        name = n
        age = a
    }
    @Override
    public int compareTo(Person other){
    // return -ve → if current should be on left
        // 0 → no change in order
        // +ve → current should be on right
        return (this.age - other.age) //asc order
    }
}
```

⇓

Natural ordering of person class.

<u>Comparator</u> → It is an interface that provides a way to define <u>custom ordering</u> of objects.

Q → Sort Person list wrs age in descending order.

```
Collections.sort(al, new AgeComparator())

class AgeComparator implements Comparator<Person>{

        @Override
        public int compare(Person p1, Person p2){
            return (p2.age - p1.age) //desc order
        }
}
```

comparator allow us to sort the list of objects without modifying the object class.