Collections Framework (PHASE 1 – DEEP FOUNDATION)

1. Collection Framework Kya Hai? (Deep Definition)

Java me:

Collections Framework = A complete architecture to store, manipulate, search, sort and process a group of objects efficiently.

Isme 3 cheeze hoti hain:

1️⃣ Interfaces

2️⃣ Classes

3️⃣ Utility Methods (`Collections` class)

2. Collection vs Collections (Very Common Interview Confusion)

Term	Meaning
Collection	Interface (List, Set, Queue ka parent)
Collections	Utility class (sort, reverse, shuffle, etc.)

Example:

```
Collections.sort(list);
```

3. Iterable vs Collection (Advanced Base Concept)

☑ Iterable (Top level)

- for-each loop isi se kaam karta hai
- Sirf ek method hota hai:

```
Iterator iterator();
```

☑ Collection (extends Iterable)

- List, Set, Queue ka parent
- Isme add(), remove(), size(), clear() jaise methods hote hain

Hierarchy:

```
Iterable
   ↑
Collection
   ↑
```

```
List    Set    Queue
```

4. Array vs Collection (Internal Difference)

Feature	Array	Collection
Size	Fixed	Dynamic
Store	Primitive + Object	Only Object
Methods	Very few	100+ methods
Sorting	Manual	Collections.sort()
Searching	Manual	contains(), indexOf()

Industry me mostly Collection use hota hai.

5. Core Interfaces of Collections (Hierarchy Explained)

```
Collection (root)
├── List
│    ├── ArrayList
│    ├── LinkedList
│    ├── Vector
│    └── Stack
├── Set
│    ├── HashSet
│    ├── LinkedHashSet
│    └── TreeSet
└── Queue
     ├── PriorityQueue
     └── ArrayDeque
Map (separate hierarchy)
├── HashMap
├── LinkedHashMap
├── TreeMap
└── Hashtable
```

6. List vs Set vs Queue vs Map (Deep Logic)

Interface	Duplicate	Order	Null	Best Use
List	✅ Allowed	✅ Yes	✅ Yes	Index-based access
Set	❌ Not allowed	❌ Mostly no	✅ One null	Unique data

Queue	✅ Allowed	✅ FIFO	✅ Yes	Task processing
Map	Key ❌	❌	Key: once	Key-Value lookup

7. Common Methods in Collection (Interview Must)

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

Example:

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

list.add(10);
list.add(20);

System.out.println(list.size());      // 2
System.out.println(list.contains(10)); // true
list.remove(0);
list.clear();
```

8. Iterator (Traversal ka Base Concept)

```
Iterator<Integer> it = list.iterator();

while(it.hasNext()) {
    System.out.println(it.next());
}
```

Agar iteration ke time structure modify karoge → `ConcurrentModificationException` aayega
(Ye ArrayList vs LinkedList me deep aayega)

ArrayList – INTERNAL WORKING (MASTER LEVEL)

1. ArrayList Kya Hota Hai? (1 Line Pro Definition)

ArrayList is a resizable (dynamic) array used to store ordered and duplicate elements.

2. ArrayList Internally Kis Par Based Hota Hai?

💡 **ArrayList internally ek NORMAL ARRAY par based hota hai.**

Actual internal code (simplified):

```
transient Object[] elementData;
```

👉 Matlab ArrayList ke andar ek array hota hai:

```
[ 10 | 20 | 30 | 40 | null | null | null ... ]
```

3. Default Capacity = 10 (Very Important Interview Question)

```
ArrayList<Integer> list = new ArrayList<>();
```

✅ Jab aap pehla element add karte ho →

➡️ Internally **10 size ka array create hota hai**

4. ArrayList Resizing Ka Formula (Most Important)

Jab ArrayList full ho jata hai →

Uska size **1.5 times** grow hota hai:

Formula:

```
newCapacity = oldCapacity + (oldCapacity / 2)
```

Example:

Current	New Capacity
10	15
15	22

22	33
33	49

5. add() Method Internally Kaise Kaam Karta Hai?

```
list.add(10);
```

Internally steps:

1️⃣ Space check hota hai

2️⃣ Agar jagah nahi → new array banata hai

3️⃣ Purana data copy hota hai

4️⃣ Naya element add hota hai

Isi wajah se kabhi-kabhi ArrayList slow ho jata hai (resize ke time)

6. Time Complexity (Interview Favourite)

Operation	Time
add()	O(1) ✅
add(index)	O(n) ❌
get()	O(1) ✅
remove(index)	O(n) ❌
search	O(n) ❌

Sabse fast operation:

✓ `get(index)`

✓ `set(index, value)`

Sabse slow:

❌ `add(0, value)`

❌ `remove(0)`

7. Memory Waste Problem in ArrayList

Agar:

Capacity = 100

Elements = 10

90 space **waste** ho jata hai

Isliye huge data ke liye **LinkedList kabhi-kabhi better hota hai**

8. remove() Method Internally Kaise Kaam Karta Hai?

```
list.remove(1);
```

Steps:

1️⃣ Index 1 ka element delete

2️⃣ Uske baad ke saare elements **left shift**

3️⃣ Last element null ho jata

Time → **O(n)**

Isliye remove slow hota hai.

9. Why ArrayList is FAST in Searching?

Because:

- Direct index access hota hai
- Array based hai

```
list.get(5000);   // Direct memory access
```

No traversal needed ✅

10. ArrayList is NOT Thread-Safe ❌

Matlab:

Multiple threads → same ArrayList →

Data corrupt ho sakta hai

Solution:

```
Collections.synchronizedList(list);
```

or

```
CopyOnWriteArrayList
```

(Ye hum advanced phase me karenge)

11. Initial Capacity Set Karna (Performance Trick)

Agar aapko pata hai ki list me 10,000 data aane wale hain:

```
ArrayList<Integer> list = new ArrayList<>(10000);
```

✅ Isse:

- Repeated resizing nahi hoga
- Performance bahut improve hoti hai

12. ArrayList Kab Use Karein?

✅ Jab:

- Searching fast chahiye
- Data mostly read-only ho
- Frequent get(index) use hota ho
- Random access required ho

❌ Jab:

- Frequent insertion/deletion ho
- Middle se delete karna padta ho
- Memory optimize karna ho

13. Most Dangerous Interview Trap ❌

Q: **ArrayList me remove fast hota hai ya LinkedList me?**

✅ Answer:

- **ArrayList → searching fast**
- **LinkedList → insertion/deletion fast**

LinkedList – INTERNAL WORKING (MASTER LEVEL)

1. LinkedList Kya Hota Hai? (Pro Definition)

LinkedList is a collection where data is stored in nodes, and each node contains a value and reference to next & previous node.

Matlab:

- Data contiguous nahi hota (jaise array)
- Har element alag memory location par hota hai
- Pointers (references) se connected hota hai

2. LinkedList Internally Kis Par Based Hota Hai?

Java ki LinkedList:

✅ **Doubly Linked List** hoti hai

Har node ke andar:

```
class Node {
    Object data;
    Node next;
    Node prev;
}
```

Structure:

```
null ← [10] ⇄ [20] ⇄ [30] → null
```

3. LinkedList Memory Structure (Very Important)

Har element ke saath:

- 1 data
- 1 next reference
- 1 previous reference

Isliye:

❌ LinkedList memory zyada consume karta hai

✅ Par performance insertion/deletion me fast hoti hai

4. add() LinkedList me Kaise Kaam Karta Hai?

```
list.add(10);
```

Steps:

1️⃣ New node create hota hai

2️⃣ Last node ke next ko new node point karaya jata hai

3️⃣ New node ka prev last ko point karta hai

4️⃣ Tail update hoti hai

⏱️ Time → **O(1)** ✅

5. get(index) LinkedList me Slow Kyun Hota Hai?

```
list.get(5000);
```

Steps:

1️⃣ Head se traversal start

2️⃣ Next → Next → Next → ...

3️⃣ Jab tak index nahi milta

4️⃣ Tab value return

⏱️ Time → **O(n)** ❌

6. remove(index) LinkedList me Fast Kyun Hota Hai?

Jaise hi node mil jata:

- Prev ka next → next
- Next ka prev → prev

No shifting ✅

⏱️ Time → **O(1)** ✅ (after reaching node)

7. LinkedList Time Complexity Table

Operation	Time
add()	O(1) ✅
add(0, value)	O(1) ✅
get(index)	O(n) ❌
remove(index)	O(1) ✅
search	O(n) ❌

8. LinkedList vs ArrayList (Deep Comparison)

Feature	ArrayList	LinkedList
Internal	Dynamic Array	Doubly Linked List
get()	O(1) ✅	O(n) ❌
add at end	O(1) ✅	O(1) ✅
add at mid	O(n) ❌	O(1) ✅
remove	Slow	Fast
Memory	Less	More
Cache friendly	✅	❌

9. LinkedList Also Implements Queue & Deque

```
LinkedList<Integer> list = new LinkedList<>();
list.addFirst(10);
list.addLast(20);
list.removeFirst();
list.removeLast();
```

Isliye LinkedList:

✅ List bhi hai

✅ Queue bhi hai

✅ Deque bhi hai

10. LinkedList Kab Use Karein?

✅ Jab:

- Frequent insertion/deletion ho
- Queue/Deque banana ho
- FIFO/LIFO use case ho

❌ Jab:

- Fast searching chahiye
- Random access chahiye
- Memory optimize karni ho

11. MOST DANGEROUS INTERVIEW TRAPS

❓ Q: ArrayList fast hoti hai ya LinkedList?

✅ Correct Answer:

- Searching → ArrayList
- Insertion/Deletion → LinkedList

Q: LinkedList me get(0) fast hota hai?

✅ Yes → head ke paas hota hai

✅ O(1)

Q: LinkedList thread-safe hoti hai?

❌ No

Same problem as ArrayList

12. Real Project Examples

✓ Browser back/forward

✓ Undo/redo

✓ Music playlist

✓ Job queues

✓ Chat message queue

Vector & Stack + Fail-Fast vs Fail-Safe (MASTER LEVEL)

1. Vector Kya Hota Hai?

Vector ek legacy class hai jo ArrayList jaisi hi hoti hai, BUT thread-safe hoti hai.

Matlab:

- Dynamic array ✅
- Duplicate allowed ✅
- Order maintained ✅
- **Synchronized (Thread-safe)** ✅

2. Vector vs ArrayList (Core Difference)

Feature	ArrayList	Vector
Thread-safe	❌ No	✅ Yes
Synchronization	❌ No	✅ Full
Performance	✅ Fast	❌ Slow
Legacy	❌ No	✅ Yes

Interview Line:

“Vector is thread-safe but slow due to synchronization overhead.”

3. Vector Growth Formula (VERY IMPORTANT)

Vector ka resizing **100%** hota hai (double):

```
newCapacity = oldCapacity * 2
```

ArrayList me:

```
1.5x
```

Isliye:

✅ Vector zyada memory khata hai

✅ Aur zyada slow hota hai

4. Vector Kab Use Karein?

✅ Jab:

- Old legacy project ho
- Thread safety chahiye ho
- Java 1.0 / 1.1 codebase ho

❌ Modern projects me:

- **Never recommended**
- Iski jagah:

```
Collections.synchronizedList(new ArrayList<>());
```

ya

```
CopyOnWriteArrayList
```

STACK (LIFO Structure)

5. Stack Kya Hota Hai?

Stack follows LIFO – Last In, First Out

Example:

- Plate stack
- Undo/Redo
- Browser back button

6. Stack is Child of Vector (Big Interview Surprise)

```
public class Stack<E> extends Vector<E>
```

Matlab:

- ✅ Stack bhi **synchronized** hota hai
- ✅ Stack bhi **slow** hota hai

7. Stack Main Methods

Method	Meaning
push()	Add
pop()	Remove
peek()	Top element
empty()	Check
search()	Position

Example:

```
Stack<Integer> s = new Stack<>();

s.push(10);
s.push(20);
System.out.println(s.pop());   // 20
System.out.println(s.peek());  // 10
```

8. Why Stack is NOT Recommended Today?

Because:

- ✅ It is synchronized → slow
- ✅ Based on Vector → legacy
- ✅ Modern replacement:

```
Deque<Integer> stack = new ArrayDeque<>();
```

Fail-Fast vs Fail-Safe (MOST IMPORTANT INTERVIEW TOPIC)

9. Fail-Fast Kya Hota Hai?

Fail-Fast iterator turant exception throw karta hai agar collection ko modify kiya jaye iteration ke time.

Exception:
`ConcurrentModificationException`

☑️ Fail-Fast Example (ArrayList)

```
ArrayList<Integer> list = new ArrayList<>();
list.add(10);
list.add(20);

Iterator<Integer> it = list.iterator();

while(it.hasNext()){
    Integer x = it.next();
    list.add(30);   // ❌ Modify during iteration
}
```

Output:
`ConcurrentModificationException`

10. Fail-Safe Kya Hota Hai?

Fail-Safe iterator copy par kaam karta hai, isliye original modify karne par exception nahi aata.

Fail-Safe Example (CopyOnWriteArrayList)

```
CopyOnWriteArrayList<Integer> list = new CopyOnWriteArrayList<>();
list.add(10);
list.add(20);

Iterator<Integer> it = list.iterator();

while(it.hasNext()){
    Integer x = it.next();
    list.add(30);   // ✅ Safe
```

}

✅ No exception
✅ Iteration old copy par hogi

11. Fail-Fast vs Fail-Safe (Table)

Feature	Fail-Fast	Fail-Safe
Exception	✅ Yes	❌ No
Collection type	Original	Copy
Performance	Fast	Slow
Memory	Less	More
Example	ArrayList	CopyOnWriteArrayList
Thread-safe	❌ No	✅ Yes

12. Which Collections Are Fail-Fast?

✅ Fail-Fast:

- ArrayList
- LinkedList
- HashSet
- HashMap

✅ Fail-Safe:

- CopyOnWriteArrayList
- ConcurrentHashMap

QUICK INTERVIEW BULLETS

✓ Vector = synchronized ArrayList
✓ Vector slower than ArrayList
✓ Stack = child of Vector
✓ Stack is LIFO
✓ Fail-Fast → exception
✓ Fail-Safe → no exception
✓ ConcurrentModificationException → Fail-Fast sign

Real Interview Question

❓ **Why CopyOnWriteArrayList is Fail-Safe?**

✅ Answer:

Because it creates a new copy of underlying array on every modification, so iterator works on old snapshot.

SET INTERFACE – DEEP FOUNDATION

1. Set Kya Hota Hai?

Set ek aisi collection hai jo duplicate elements allow nahi karti.

✅ Duplicate ❌

✅ Order mostly ❌

✅ One null allowed (HashSet me)

Example:

```
Set<Integer> set = new HashSet<>();
set.add(10);
set.add(20);
set.add(10);

System.out.println(set);  // [10, 20]
```

2. Set Implementations

Class	Duplicate	Order	Internal Structure
HashSet	❌	❌	HashTable
LinkedHashSet	❌	✅	HashTable + LinkedList
TreeSet	❌	✅ Sorted	Red-Black Tree

HASHSET – INTERNAL WORKING (MASTER LEVEL)

3. HashSet Internally Kis Par Based Hota Hai?

HashSet internally → HashMap par based hota hai

```
public class HashSet<E> {
    private transient HashMap<E,Object> map;
```

```
}
```

✅ HashSet ka har element → **HashMap ki key hota hai**
✅ Value ek dummy object hota hai

4. HashSet me add() Kaise Kaam Karta Hai?

```
set.add("A");
```

Internally yeh hota hai:

```
map.put("A", DUMMY_OBJECT);
```

5. Duplicate Kaise Detect Hota Hai? (CORE LOGIC)

HashSet duplicate detect karta hai **2 methods se**:

1️⃣ `hashCode()`
2️⃣ `equals()`

Sequence:

1. Pehle `hashCode()` call hota hai
2. Same bucket milta hai kya?
3. Agar same bucket → `equals()` call hota hai
4. Agar equals true → ❌ duplicate → add nahi hoga
5. Agar equals false → ✅ add ho jayega

hashCode() & equals() – INTERVIEW KILLER

6. Agar equals override kiya aur hashCode nahi kiya to?

❌ Duplicate aa sakta hai
❌ HashSet break ho jata hai
❌ Interview me direct reject waala question

7. Golden Rule (VERY IMPORTANT)

✅ **Agar equals override karte ho to hashCode bhi override karna hi padega.**

8. Real Custom Object Example (Most Important)

❌ WITHOUT equals & hashCode

```
class Student {
    int id;
    String name;
```

```
    Student(int id, String name){
        this.id = id;
        this.name = name;
    }
}

HashSet<Student> set = new HashSet<>();

set.add(new Student(1,"A"));
set.add(new Student(1,"A"));

System.out.println(set.size()); // ❌ Output = 2 (Duplicate allowed!)
```

WITH equals & hashCode (Correct Way)

```
class Student {
    int id;
    String name;

    Student(int id, String name){
        this.id = id;
        this.name = name;
    }

    @Override
    public boolean equals(Object obj){
        Student s = (Student) obj;
        return this.id == s.id;
    }

    @Override
    public int hashCode(){
        return id;
    }
}

HashSet<Student> set = new HashSet<>();
set.add(new Student(1,"A"));
set.add(new Student(1,"A"));

System.out.println(set.size()); // ✅ Output = 1
```

9. HashSet Time Complexity

Operation	Time
add()	O(1) ✅
remove()	O(1) ✅
contains()	O(1) ✅

⚠️ Worst case me agar sab ek bucket me gir gaye → O(n)

10. HashSet Allow NULL?

✅ YES — **Only one null allowed**

11. HashSet Kab Use Karein?

✅ Jab:

- Duplicate data allow nahi karna
- Searching fastest chahiye
- Order matter na karta ho

❌ Jab:

- Order maintain karna ho
- Sorted data chahiye

LINKEDHASHSET (Quick Deep)

✅ HashSet jaisa hi hota hai
✅ BUT **insertion order preserve karta hai**
Internally:

```
HashTable + Doubly Linked List
```

Use:
✓ Cache
✓ LRU
✓ History maintain karni ho

TREESET (Quick Intro – Deep baad me)

✅ Sorted order
✅ Red-Black tree
✅ Slower than HashSet
✅ No null allowed
Example:

```
TreeSet<Integer> ts = new TreeSet<>();
ts.add(40);
ts.add(10);
ts.add(20);

System.out.println(ts); // [10, 20, 40]
```

MOST DANGEROUS INTERVIEW QUESTIONS

❓ HashSet duplicate kaise detect karta hai?
✅ **hashCode() + equals()**
❓ Agar equals override kiya aur hashCode nahi kiya?
✅ Duplicate aa jayega
❓ HashSet internally kis par based hai?
✅ HashMap
❓ HashSet sorted hota hai?
✅ ❌ No

1-MIN REVISION

✓ Set = unique collection
✓ HashSet = fastest
✓ Internally = HashMap
✓ Duplicate detect = hashCode + equals
✓ One null allowed
✓ Order not guaranteed

TREESET — SORTED SET (MASTER LEVEL)

1. TreeSet Kya Hota Hai? (Pro Definition)

TreeSet ek aisa Set hai jo elements ko hamesha SORTED order me store karta hai.

✅ Duplicate ❌
✅ Sorted order ✅
✅ Null ❌ (Java 7+ me strictly not allowed)
✅ Slower than HashSet
✅ Based on **Red-Black Tree**

2. TreeSet Internally Kis Par Based Hota Hai?

TreeSet internally → TreeMap par based hota hai

```
public class TreeSet<E> {
    private transient TreeMap<E,Object> m;
}
```

Aur **TreeMap → Red-Black Tree** par based hota hai.

3. Red-Black Tree (Interview Level Logic)

Red-Black Tree ek **self-balancing Binary Search Tree (BST)** hota hai:

✅ Har insert ke baad tree auto-balance hota hai

✅ Height hamesha controlled hoti hai

✅ Search, Insert, Delete → **O(log n)**

Isliye:

- TreeSet slow hai compared to HashSet (O(1))
- But TreeSet sorted output deta hai ✅

4. TreeSet Example (Default Sorting)

```
TreeSet<Integer> ts = new TreeSet<>();

ts.add(50);
ts.add(10);
ts.add(40);
ts.add(20);

System.out.println(ts);  // [10, 20, 40, 50]
```

Automatically ascending order me sort ho gaya

5. TreeSet Duplicate Kaise Detect Karta Hai?

⚠️ TreeSet **hashCode() + equals() use nahi karta**

TreeSet duplicate detect karta hai:

✅ **compareTo() / compare() == 0 → duplicate**

Matlab:

- Agar comparison result = 0
- To TreeSet use duplicate samajhta hai

COMPARABLE vs COMPARATOR (VERY VERY IMPORTANT)

Ye topic **almost har interview me aata hai**.

6. Comparable Kya Hota Hai?

Comparable ek interface hai jo object ke andar hi sorting logic define karta hai.

Method:

```
public int compareTo(Object o)
```

Comparable Example

```
class Student implements Comparable<Student> {
    int id;
    String name;
    Student(int id, String name){
        this.id = id;
        this.name = name;
    }

    public int compareTo(Student s){
        return this.id - s.id;   // ascending by id
    }
}

TreeSet<Student> ts = new TreeSet<>();
ts.add(new Student(3,"A"));
ts.add(new Student(1,"B"));
ts.add(new Student(2,"C"));

for(Student s : ts)
    System.out.println(s.id);
```

Output:

```
1
2
3
```

7. Comparator Kya Hota Hai?

Comparator external sorting logic deta hai (class ke bahar).

Method:

```
public int compare(Object o1, Object o2)
```

Comparator Example (Sort by Name)

```
class NameComparator implements Comparator<Student> {
    public int compare(Student s1, Student s2){
        return s1.name.compareTo(s2.name);
    }
}
```

Use in TreeSet:

```
TreeSet<Student> ts = new TreeSet<>(new NameComparator());
```

8. Comparable vs Comparator (INTERVIEW TABLE)

Feature	Comparable	Comparator
Package	java.lang	java.util
Method	compareTo()	compare()
Where logic	Same class	Separate class
How many	Only 1	Multiple
Modify class?	✅ Yes	❌ No

9. Very Dangerous Interview Trap ❌

❓ Agar TreeSet me custom object add kiya aur Comparable/Comparator nahi diya?

Runtime Exception aayega:

`ClassCastException`

Kyun?

TreeSet ko pata hi nahi hoga **compare kaise karna hai**

10. TreeSet Null Kyun Allow Nahi Karta?

Because:

- Sorting ke liye comparison hota hai
- Null compare nahi ho sakta

Isliye:

`NullPointerException`

11. TreeSet Time Complexity

Operation	Time
add()	O(log n)
remove()	O(log n)

contains()	O(log n)

HashSet:

✅ Sab O(1)

TreeSet:

✅ Sorted but slower

12. TreeSet Kab Use Karein?

✅ Jab:

- Sorted data chahiye
- Unique + ordered elements chahiye
- Ranking system ho
- Leaderboard
- Marks list
- Price sorting

❌ Jab:

- Only fast searching chahiye
- Sorting ka requirement nahi ho

MOST ASKED INTERVIEW QUESTIONS

❓ TreeSet HashSet se slow kyun hota hai?

✅ Because TreeSet → Red-Black Tree → O(log n)

❓ TreeSet duplicate kaise remove karta hai?

✅ compareTo() == 0 par

❓ TreeSet me null kyun allowed nahi?

✅ Because comparison me NPE aata hai

❓ Comparable aur Comparator me core difference?

✅ One internal, one external sorting

1-MIN REVISION

✓ TreeSet = Sorted Set

✓ Based on TreeMap

✓ TreeMap = Red-Black Tree

✓ Duplicate detect = compareTo / compare

✓ Null ❌

✓ Custom object ke liye Comparable/Comparator mandatory

QUEUE INTERFACE – MASTER LEVEL

1. Queue Kya Hota Hai? (Exact Definition)

Queue ek aisi collection hai jo FIFO (First In First Out) principle par kaam karti hai.

Real-life examples:

✓ Ticket counter
✓ Printer queue
✓ Call center
✓ CPU Scheduling

2. Queue Hierarchy (Important)

```
Queue (interface)
   ↑
Deque (interface)
   ↑
ArrayDeque (class)

PriorityQueue (class) → direct Queue implement karta hai
```

3. Queue vs List (Interview Difference)

Feature	Queue	List
Access	FIFO	Index based
get(index)	❌	✅
Duplicate	✅	✅
Order	FIFO	Insertion order

4. Important Queue Methods (Interview Must)

Method	If Fail
add()	Exception
offer()	false
remove()	Exception
poll()	null
element()	Exception
peek()	null

PRIORITYQUEUE — HEAP BASED QUEUE

5. PriorityQueue Kya Hota Hai?

PriorityQueue ek aisi queue hai jahan FIFO follow nahi hota, balki PRIORITY follow hoti hai.

Example:

- Emergency patient pehle
- High priority job pehle
- CPU task scheduling

6. PriorityQueue Internally Kis Par Based Hota Hai?

🔥 **PriorityQueue internally → MIN HEAP par based hota hai**

Matlab:

✅ Sabse chhota element hamesha top par hota hai

✅ Complete Binary Tree ka structure hota hai

✅ Array ke andar store hota hai

7. PriorityQueue Example (Default Min Heap)

```
PriorityQueue<Integer> pq = new PriorityQueue<>();

pq.add(40);
pq.add(10);
pq.add(30);
pq.add(20);

System.out.println(pq);     // Output not sorted
System.out.println(pq.poll()); // ✅ 10 (smallest removed first)
```

Important:

⚠️ `System.out.println(pq)` sorted print nahi hota

✅ But `poll()` hamesha smallest deta hai

8. Max Heap PriorityQueue (Custom Comparator)

By default → Min Heap

Max Heap banane ke liye:

```
PriorityQueue<Integer> pq =
    new PriorityQueue<>(Collections.reverseOrder());
```

9. PriorityQueue Custom Object (Very Important)

```
class Job {
    int priority;
    String name;
```

```
    Job(int p, String n){
        priority = p;
        name = n;
    }
}
```

Comparator:

```
PriorityQueue<Job> pq = new PriorityQueue<>(
    (a, b) -> a.priority - b.priority
);
```

10. PriorityQueue Time Complexity

Operation	Time
add()	O(log n)
poll()	O(log n)
peek()	O(1)

11. PriorityQueue vs TreeSet

Feature	PriorityQueue	TreeSet
Duplicate	✅ Allowed	❌ Not allowed
Ordering	Only top sorted	Full sorted
Internal	Heap	Red-Black Tree
get min	O(1)	O(log n)
Iteration sorted?	❌ No	✅ Yes

DEQUE (Double Ended Queue)

☑ 12. Deque Kya Hota Hai?

Deque ek aisi queue hai jisme insertion aur deletion dono ends se hota hai.

Matlab:

✅ Front se bhi

✅ Rear se bhi

☑ 13. Deque se Stack + Queue dono ban sakte hain

```
Deque<Integer> dq = new ArrayDeque<>();

// Stack
```

```
dq.push(10);
dq.pop();

// Queue
dq.offer(20);
dq.poll();
```

ARRAYDEQUE — MODERN STACK & QUEUE

14. ArrayDeque Kya Hota Hai?

ArrayDeque ek resizable circular array based Deque hai.

✅ Very fast
✅ No synchronization
✅ Stack aur Queue dono ka best replacement
✅ Stack class se **beat karta hai performance me**

15. ArrayDeque Internal Working

- Circular array
- Head & Tail pointers
- No shifting
- Isliye:
  - ✅ addFirst(), addLast() → O(1)

16. ArrayDeque Restrictions

❌ Null not allowed
❌ Thread-safe nahi hai

17. Deque vs Stack (Interview Trap)

Stack	Deque
Legacy	Modern
Synchronized	❌ No
Slow	Fast
Based on Vector	Based on circular array

✅ Interview answer:
“Stack is legacy and slow. We should use Deque/ArrayDeque instead.”

MOST DANGEROUS INTERVIEW QUESTIONS (QUEUE)

❓ PriorityQueue sorted hota hai?
✅ ❌ Fully sorted nahi, sirf head sorted hota hai
❓ PriorityQueue null allow karta hai?
✅ ❌ No
❓ PriorityQueue duplicate allow karta hai?
✅ ✅ Yes
❓ Best Stack replacement kaunsa hai?
✅ ✅ ArrayDeque

1-MIN REVISION

✓ Queue = FIFO
✓ PriorityQueue = Heap based
✓ poll() = highest priority element
✓ Deque = double-ended queue
✓ ArrayDeque = fastest stack & queue
✓ Stack = legacy ❌

MAP INTERFACE – DEEP FOUNDATION

1. Map Kya Hota Hai?

Map ek aisi collection hai jo data ko key–value pair me store karti hai.
Example:

```
RollNo → Student
ID → Employee
Username → Password
```

2. Map Collection ka Part Kyun Nahi Hai?

Because:

- Collection sirf **values** store karta hai
- Map **key + value** store karta hai
  Isliye Map ka **separate hierarchy** hai.

3. Map Implementations

Class	Duplicate Key	Order	Thread-safe
HashMap	❌	❌	❌
LinkedHashMap	❌	✅	❌
TreeMap	❌	✅ Sorted	❌
Hashtable	❌	❌	✅
ConcurrentHashMap	❌	❌	✅

HASHMAP – INTERNAL WORKING (INTERVIEW DESTROYER)

4. HashMap Internally Kis Par Based Hota Hai?

HashMap internally → Array + LinkedList + (Java 8 se) Red-Black Tree

Structure:

```
Bucket Array (size 16 default)

[0] → null
[1] → (Node → Node → Node)
[2] → null
[3] → Tree (if collision zyada ho)
```

5. HashMap Default Capacity & Load Factor (VERY IMPORTANT)

```
new HashMap<>();
```

Term	Value
Default Capacity	**16**
Load Factor	**0.75**

Matlab:

Jab:

```
16 × 0.75 = 12
```

12 entries ke baad **rehashing (resize)** hoga.

6. put() Method Internal Working (STEP BY STEP)

```
map.put("A", 100);
```

Steps:

1️⃣ `"A".hashCode()` calculate hota hai
2️⃣ Hash ko compress kiya jata hai
3️⃣ Index nikala jata hai:

```
index = hash & (n - 1)
```

4️⃣ Us bucket par:

- Agar empty → new Node add
- Agar node already hai → collision handle

7. Collision Kya Hota Hai?

Jab **2 keys ka index same aa jaye** → collision
Example:

```
"A" → index 3
"B" → index 3
```

Dono same bucket me chale jaate hain.

8. Collision Handling in HashMap

Java 7 tak:

➡️ **LinkedList**

```
( Key1 → Key2 → Key3 )
```

Java 8 se:

Agar bucket me:

```
> 8 nodes ho gaye
```

➡️ **LinkedList → Red-Black Tree**
➡️ Is process ko kehte hain:
TREEIFICATION

9. Treeification Conditions (Very Important)

Tree banega **tabhi** jab:

✅ Bucket size ≥ 8
✅ Total capacity ≥ 64

Agar capacity 64 se kam hai →
🔁 Rehash hota hai, tree nahi banta

10. get() Method Internal Working

```
map.get("A");
```

Steps:

1️⃣ Key ka hash banta hai
2️⃣ Index calculate hota hai
3️⃣ Us bucket me:

- LinkedList → traversal
- Tree → binary search

4️⃣ equals() ka use karke key match hoti hai
5️⃣ Value return hoti hai

11. Time Complexity of HashMap

Case	Time
Best case	O(1) ✅
With collision (LinkedList)	O(n) ❌
With Tree (Java 8+)	O(log n) ✅

12. HashMap Duplicate Key Kaise Handle Karta Hai?

```
map.put("A", 10);
map.put("A", 20);
```

✅ Old value overwrite ho jaati hai
✅ New value replace ho jaati hai

Because:

- `hashCode()` same
- `equals()` true
  → Duplicate key detect

13. HashMap NULL Handling (Most Asked)

Item	Allowed?
1 null key	✅ Yes
Multiple null values	✅ Yes

Why only one null key?

Because:

- Key unique hoti hai
- Null ka hash group ek hi hota hai

14. equals() & hashCode() in HashMap (Same As Set)

HashMap bhi **duplicate key detect karta hai** using:

✅ `hashCode()`
✅ `equals()`

Same rule:
Equals override → HashCode override mandatory

INTERVIEW TRAPS (VERY DANGEROUS)

❓ HashMap thread-safe hota hai?
✅ ❌ NO
❓ Hashtable aur HashMap me difference?
✅ Hashtable synchronized hota hai
✅ HashMap fast hota hai
❓ HashMap resize kab hota hai?
✅ Jab:

```
size > capacity × 0.75
```

1-MIN MASTER REVISION

✓ HashMap = key-value store
✓ Based on Array + LinkedList + Tree
✓ Default capacity = 16
✓ Load factor = 0.75
✓ Collision handled by LinkedList / Tree
✓ Tree banta hai when bucket ≥ 8
✓ One null key allowed
✓ Duplicate key overwrite hoti hai
✓ Best time = O(1)

LINKEDHASHMAP — ORDER + CACHE

1. LinkedHashMap Kya Hota Hai?

LinkedHashMap ek aisa HashMap hai jo insertion order ko remember rakhta hai.

HashMap:
❌ Koi order nahi
LinkedHashMap:
✅ Order maintain hota hai

2. LinkedHashMap Internally Kaise Kaam Karta Hai?

Internally:

✅ **HashMap + Doubly Linked List**

Structure:

```
Bucket Array
     ↓
 Node → Node → Node (insertion order)
```

Isliye:

- Fast like HashMap ✅
- Order bhi maintain ✅

3. LinkedHashMap Example

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

map.put(3, "C");
map.put(1, "A");
map.put(2, "B");

System.out.println(map);
```

✅ Output:

```
{3=C, 1=A, 2=B}
```

➡️ Exactly wahi order jis order me insert kiya

4. LinkedHashMap Access Order (Very Important)

LinkedHashMap ke 2 modes hote hain:

```
LinkedHashMap(int capacity, float loadFactor, boolean accessOrder)
```

Mode	Meaning
false (default)	Insertion Order
true	Access Order

5. LinkedHashMap se LRU Cache (INTERVIEW FAVORITE)

☑ **LRU = Least Recently Used**

```
class LRUCache<K, V> extends LinkedHashMap<K, V> {

    private int capacity;
```

```
    LRUCache(int capacity) {
        super(capacity, 0.75f, true); // accessOrder = true
        this.capacity = capacity;
    }

    protected boolean removeEldestEntry(Map.Entry<K, V> eldest) {
        return size() > capacity;
    }
}
```

✅ Ye automatic oldest unused element ko delete karta rahega ✅
Real use:

- Browser cache
- Database cache
- Redis logic
- Image caching

TREEMAP — SORTED MAP

6. TreeMap Kya Hota Hai?

TreeMap ek aisa Map hai jo keys ko hamesha sorted order me rakhta hai.

✅ Sorted by key
✅ Duplicate key ❌
✅ Thread-safe ❌
✅ Null key ❌ (single null Java 7 tak tha)
✅ Based on **Red-Black Tree**

7. TreeMap Example

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

map.put(30, "C");
map.put(10, "A");
map.put(20, "B");

System.out.println(map);
```

✅ Output:

```
{10=A, 20=B, 30=C}
```

8. TreeMap vs HashMap

Feature	HashMap	TreeMap
Order	❌	✅ Sorted
Internal	Array + LL + Tree	Red-Black Tree
Time	O(1)	O(log n)
Null key	✅ 1	❌
Use	Fast access	Sorted data

9. Custom Object in TreeMap (Comparator)

Same rule:

✅ Comparable / Comparator mandatory

```
TreeMap<Student, Integer> map = new TreeMap<>(
    (s1, s2) -> s1.id - s2.id
);
```

HASHTABLE vs CONCURRENTHASHMAP

10. Hashtable (Legacy)

❌ Slow
❌ Full synchronization
❌ No null key/value
✅ Thread-safe

11. ConcurrentHashMap (Modern)

✅ High performance
✅ Segmented locking
✅ Thread-safe
✅ Null key/value ❌

Used in:

- Multithreaded apps
- Web servers
- Spring Boot internally

1-MIN MASTER REVISION

✓ LinkedHashMap = insertion + access order
✓ HashMap + Doubly Linked List

✓ TreeMap = Sorted Map
✓ TreeMap = Red-Black Tree
✓ LRU Cache = LinkedHashMap
✓ ConcurrentHashMap = Multithreading Map

1. Iterator Kya Hota Hai? (Quick Recall)

Iterator ek object hota hai jisse hum Collection ke elements ko ek–ek karke traverse karte hain.

```
Iterator<Integer> it = list.iterator();

while(it.hasNext()) {
    System.out.println(it.next());
}
```

2. FAIL-FAST Iterator Kya Hota Hai?

Fail-Fast iterator turant exception throw kar deta hai agar iteration ke dauraan collection me structural modification ho jaye.

👉 Ye exception hota hai:
`ConcurrentModificationException`

Fail-Fast Konse Collections Hote Hain?

✅ ArrayList
✅ HashMap
✅ HashSet
✅ TreeSet
✅ LinkedList

Matlab:
java.util package ke mostly saare collections Fail-Fast hote hain

Fail-Fast Example (IMPORTANT)

```
ArrayList<Integer> list = new ArrayList<>();
list.add(10);
list.add(20);
list.add(30);

Iterator<Integer> it = list.iterator();
while(it.hasNext()) {
```

```
    Integer x = it.next();
    if(x == 20) {
        list.remove(x);  // ❌ Direct modification
    }
}
```

❌ Output:
```
Exception in thread "main" java.util.ConcurrentModificationException
```

Fail-Fast Internally Kaise Kaam Karta Hai?

Har collection ke andar 2 variables hote hain:
```
int modCount;        // actual modification count
int expectedModCount; // iterator ka copy
```

- Jaise hi collection modify hoti hai:

```
modCount++
```

- Iterator compare karta hai:

```
modCount != expectedModCount
```

➡️ Turant **ConcurrentModificationException**

3. FAIL-SAFE Iterator Kya Hota Hai?

Fail-Safe iterator collection ke CLONE par iterate karta hai, isliye original collection modify hone par bhi exception nahi deta.

Fail-Safe Konse Collections Hote Hain?

✅ CopyOnWriteArrayList
✅ CopyOnWriteArraySet
✅ ConcurrentHashMap

Matlab:
java.util.concurrent package ke collections Fail-Safe hote hain

Fail-Safe Example

```
CopyOnWriteArrayList<Integer> list = new CopyOnWriteArrayList<>();
list.add(10);
list.add(20);
list.add(30);

Iterator<Integer> it = list.iterator();
```

```
while(it.hasNext()) {
    Integer x = it.next();
    if(x == 20) {
        list.remove(x);  // ✅ Allowed
    }
}
System.out.println(list);
```

✅ Output:
`[10, 30]`

✅ No Exception
✅ Safe in multithreading

Fail-Safe Internally Kaise Kaam Karta Hai?

- Iterator ko **original list nahi milti**
- Balki:

`Collection ka snapshot / clone milta hai`

Isliye:
✅ Original modify karo → koi effect nahi
✅ Iterator safe rehta hai

4. Fail-Fast vs Fail-Safe (INTERVIEW TABLE)

Feature	Fail-Fast	Fail-Safe
Exception	✅ Yes	❌ No
Data source	Original Collection	Clone/Snapshot
Performance	✅ Fast	❌ Slow
Memory	✅ Less	❌ More
Thread-safe	❌ No	✅ Yes
Package	java.util	java.util.concurrent

5. ConcurrentModificationException Kab Aata Hai?

✅ Iterator ke dauraan agar:

- add()
- remove()

- clear()

direct collection par call kiya jaye

6. Fail-Fast me Safe Removal Ka Sahi Tarika ☑

Agar exception avoid karni ho to:

```
Iterator<Integer> it = list.iterator();

while(it.hasNext()) {
    Integer x = it.next();
    if(x == 20) {
        it.remove();  // ✅ Safe removal
    }
}
```

✅ Ye allowed hai
❌ `list.remove()` allowed nahi

7. Real-World Scenario

❓ Web server me agar 100 user ek hi list access karein?

- ❌ ArrayList → Crash possibility
- ✅ CopyOnWriteArrayList → Safe

1-MIN MASTER REVISION

✓ Fail-Fast → Exception deta hai
✓ Fail-Safe → Exception nahi deta
✓ Fail-Fast → Original collection
✓ Fail-Safe → Clone collection
✓ Fail-Fast → Fast but unsafe
✓ Fail-Safe → Safe but memory heavy
✓ ConcurrentModificationException → Fail-Fast ka signal

MOST ASKED INTERVIEW QUESTIONS

❓ Fail-Fast aur Fail-Safe me core difference?
✅ Exception + original vs clone
❓ CopyOnWriteArrayList safe kyun hoti hai?
✅ Kyunki wo clone par iterate karti hai

❓ Iterator ke dauraan direct list.remove() kyun dangerous hai?
✅ Because modCount mismatch ho jata hai

1. Collections Class Kya Hoti Hai?

`Collections` ek utility class hai jisme sirf static methods hote hain jo collections par operations perform karti hai.

⚠️ Dhyan rahe:

- `Collection` → interface hai
- `Collections` → class hai (with static methods)

2. sort() – LIST KO SORT KARNA

▶️ Ascending Order

```
ArrayList<Integer> list = new ArrayList<>();
list.add(40);
list.add(10);
list.add(30);
list.add(20);

Collections.sort(list);

System.out.println(list);
```

✅ Output:

```
[10, 20, 30, 40]
```

Descending Order (Comparator ke saath)

```
Collections.sort(list, Collections.reverseOrder());
System.out.println(list);
```

✅ Output:

```
[40, 30, 20, 10]
```

3. binarySearch() – FAST SEARCH (O(log n))

⚠️ **Pre-condition:** List pehle se sorted honi chahiye ❗

```
Collections.sort(list);

int index = Collections.binarySearch(list, 30);
```

```
System.out.println(index);
```

✅ Output:
2

❌ Agar list sorted nahi ho:
→ **Galat index milega (bug)**

4. reverse() – LIST KO ULTA KARNA

```
Collections.reverse(list);
System.out.println(list);
```

✅ Output:
```
[40, 30, 20, 10]
```

5. shuffle() – RANDOM ORDER (INTERVIEW USE)

```
Collections.shuffle(list);
System.out.println(list);
```

✅ Output (random):
```
[20, 40, 10, 30]
```

👉 Use case:

- Ludo cards
- Quiz random questions
- Game cards shuffle

6. min() & max()

```
System.out.println(Collections.min(list));
System.out.println(Collections.max(list));
```

✅ Output:
```
10
40
```

7. frequency() – COUNT OF ELEMENT

```
ArrayList<Integer> list = new ArrayList<>();
list.add(10);
list.add(20);
list.add(10);
list.add(10);

System.out.println(Collections.frequency(list, 10));
```

✅ Output:

```
3
```

8. swap() – 2 ELEMENTS KI POSITION BADALNA

```
Collections.swap(list, 0, 2);
System.out.println(list);
```

9. disjoint() – SAME ELEMENT CHECK

```
ArrayList<Integer> a = new ArrayList<>(List.of(1,2,3));
ArrayList<Integer> b = new ArrayList<>(List.of(4,5,6));

System.out.println(Collections.disjoint(a, b));
```

✅ Output:

```
true
```

👉 true = koi common element nahi
👉 false = koi common element hai

10. unmodifiableList() – READ-ONLY COLLECTION

```
List<Integer> list2 = Collections.unmodifiableList(list);

list2.add(100);   // ❌ Exception
```

❌ Output:

```
UnsupportedOperationException
```

✅ Use:

- Secure data
- Read-only APIs

11. synchronizedList() – THREAD-SAFE LIST

```
List<Integer> syncList =
    Collections.synchronizedList(new ArrayList<>());
```

✅ Multi-threading me safe access ke liye

12. Custom Object Sorting with Collections.sort()

Student Class

```
class Student {
    int id;
    String name;

    Student(int id, String name){
        this.id = id;
        this.name = name;
    }
}
```

Comparator by ID

```
Collections.sort(list, (s1, s2) -> s1.id - s2.id);
```

13. Collections vs Arrays (Interview Trap)

Collections	Arrays
For List	For Array
sort(List)	sort(array)
min(list)	No min
reverse(list)	No direct

MOST ASKED INTERVIEW QUESTIONS

❓ binarySearch fast kyun hota hai?

✅ Kyunki wo **divide & conquer** par kaam karta hai (O(log n))

❓ shuffle() ka real-world use?
✅ Gaming, quiz apps, lottery
❓ unmodifiableList kyu use karte hain?
✅ Data ko read-only banane ke liye

☑ 1-MIN REVISION

✓ Collections = utility class
✓ sort() = sorting
✓ binarySearch() = fast searching
✓ reverse() = list ulta
✓ shuffle() = random order
✓ min(), max() = extreme values
✓ unmodifiableList() = read-only
✓ synchronizedList() = thread-safe

1. Comparable Kya Hota Hai?

Comparable ek interface hai jo object ke andar hi sorting ka logic define karta hai.

```
public interface Comparable<T> {
    int compareTo(T obj);
}
```

✅ Sorting logic **same class ke andar** hota hai
✅ Sirf **EK TYPE** ki sorting possible

Comparable Example (Sort by ID)

```
class Student implements Comparable<Student> {
    int id;
    String name;

    Student(int id, String name){
        this.id = id;
        this.name = name;
    }

    @Override
    public int compareTo(Student s){
        return this.id - s.id;   // ascending by id
    }
```

```
}
```

Use with TreeSet or Collections.sort():

```
ArrayList<Student> list = new ArrayList<>();
list.add(new Student(3,"A"));
list.add(new Student(1,"B"));
list.add(new Student(2,"C"));

Collections.sort(list);  // Comparable auto called
```

✅ Output (by id):

```
1 2 3
```

2. Comparator Kya Hota Hai?

Comparator ek separate class hota hai jo object ke bahar sorting logic deta hai.

```
public interface Comparator<T> {
    int compare(T o1, T o2);
}
```

✅ Sorting logic **external hota hai**
✅ **Multiple sorting** possible

Comparator Example (Sort by Name)

```
class NameComparator implements Comparator<Student> {
    public int compare(Student s1, Student s2){
        return s1.name.compareTo(s2.name);
    }
}

Collections.sort(list, new NameComparator());
```

✅ Output (by name):

```
A B C
```

3. Same Class – Multiple Sorting (Comparator Power)

```
Comparator<Student> byId =
    (a, b) -> a.id - b.id;
```

```
Comparator<Student> byName =
    (a, b) -> a.name.compareTo(b.name);
```

✅ Interview line:
“Comparator se hum **runtime par multiple sorting strategies bana sakte hain**.”

4. TreeSet + Comparable vs Comparator (DANGEROUS TRAP)

☑ With Comparable

```
TreeSet<Student> ts = new TreeSet<>();
```

➡ Student class me `compareTo()` mandatory

☑ With Comparator

```
TreeSet<Student> ts = new TreeSet<>(new NameComparator());
```

➡ Student class me koi change nahi

5. Agar Comparable + Comparator dono ho?

Rule:
✅ **Comparator HAMESHA Comparable ko override karta hai**
(TreeSet, TreeMap, Collections.sort(list, comparator))

6. Return Values Ka Matlab (compare / compareTo)

Return Value	Meaning
`0`	Both equal (duplicate)
+ve	First object bada
-ve	First object chhota

Example:

```
return a.id - b.id;
```

7. Duplicate Detection in TreeSet / TreeMap

Very dangerous interview trap:
TreeSet / TreeMap duplicate detect karta hai using:
✅ `compare()` OR `compareTo()`
❌ `equals()` yahan kaam nahi karta
Matlab:

```
compareTo() == 0  → Duplicate
```

8. What If Comparable/Comparator Dono Nahi Diya?

```
TreeSet<Student> ts = new TreeSet<>();
ts.add(new Student(1,"A"));
```

❌ Runtime exception:
`ClassCastException`

✅ Interview answer:
“TreeSet ko pata hi nahi hota compare kaise karna hai.”

9. Comparable vs Comparator (FINAL INTERVIEW TABLE)

Feature	Comparable	Comparator
Package	java.lang	java.util
Method	compareTo()	compare()
Logic Location	Same class	Separate class
Sorting Types	One	Multiple
Class modification	Required	Not required
Used by default in	TreeSet, TreeMap	TreeSet, TreeMap, sort(list, c)

10. Real-World Use Case

✅ Comparable:

- Default natural order
- Employee by ID

- Student by Roll No

✅ Comparator:

- Salary wise
- Name wise
- Date wise
- Rating wise
- Price wise (E-commerce)

MOST ASKED INTERVIEW QUESTIONS (FINAL)

❓ Comparable aur Comparator me primary difference?
✅ Internal vs External sorting logic
❓ TreeSet duplicate kaise detect karta hai?
✅ `compareTo() == 0`
❓ Kya Comparator Comparable ko override karta hai?
✅ ✅ YES
❓ Kya ek class me 5 Comparator ho sakte hain?
✅ ✅ YES
❓ Agar compareTo galat likh diya to kya hoga?
✅ Sorting wrong + duplicates wrong

1-MIN FINAL REVISION

✓ Comparable → Natural order
✓ Comparator → Custom order
✓ Comparable → 1 sorting
✓ Comparator → Multiple sorting
✓ TreeSet duplicate → compare() == 0
✓ Comparator > Comparable (priority)