**✅ Core Java Collections – Advanced Interview Q&A**

**1. What is the difference between ArrayList and LinkedList?**

**Answer:**

| **Feature** | **ArrayList** | **LinkedList** |
| --- | --- | --- |
| Internal structure | Dynamic array | Doubly-linked list |
| Access time | O(1) for index-based access | O(n) traversal |
| Insertion | O(n) in worst case (resizing) | O(1) at head/tail |
| Memory | Less memory (data only) | More (data + 2 references per node) |
| Use case | Frequent reads | Frequent insertions/deletions |

**Follow-up:** When would you use LinkedList over ArrayList?

When you have frequent add/remove operations from the beginning or middle of the list.

**2. Explain fail-fast and fail-safe iterators in Java.**

**Answer:**

* **Fail-Fast:** Throw ConcurrentModificationException if collection is modified during iteration (e.g., ArrayList, HashMap).
* **Fail-Safe:** Work on a **copy** of the collection and allow safe iteration (e.g., CopyOnWriteArrayList, ConcurrentHashMap).

**3. What are the differences between HashMap, TreeMap, and LinkedHashMap?**

| **Feature** | **HashMap** | **TreeMap** | **LinkedHashMap** |
| --- | --- | --- | --- |
| Order | No order | Sorted (by keys) | Insertion/access order |
| Performance | Fastest (O(1)) | O(log n) | Slightly slower than HashMap |
| Null Keys | One null key | No null key allowed | One null key |

**4. How is hashing implemented in Java Collections?**

**Answer:**

* Hash-based collections like HashMap use hashCode() to determine the **bucket** and equals() to resolve **collisions**.
* Since Java 8, HashMap uses **linked list → balanced tree** (when entries in a bucket exceed threshold).

**Follow-up:** What happens when two keys have the same hashCode?

They go into the same bucket and are resolved using equals().

**5. How does ConcurrentHashMap work internally?**

**Answer:**

* Prior to Java 8: used **segment-based locking**.
* From Java 8: uses **bucket-level locking** via synchronized blocks and **CAS** (Compare-And-Swap).
* Supports **concurrent reads** and **limited concurrent writes**.

**6. What are the best practices for choosing a collection in Java?**

| **Use case** | **Recommended Collection** |
| --- | --- |
| Fast lookup | HashMap |
| Sorted order of keys | TreeMap |
| Maintain insertion order | LinkedHashMap |
| Unique elements | HashSet |
| Thread-safe list | CopyOnWriteArrayList |
| Bounded concurrent queue | ArrayBlockingQueue |

**7. How does equals() and hashCode() impact collections?**

**Answer:**

* Set and Map rely on **both** methods for:
  + Uniqueness (Set)
  + Key lookup (Map)
* Consistency rule:
  + If a.equals(b) then a.hashCode() == b.hashCode()

**8. What is the difference between Comparable and Comparator?**

| **Feature** | **Comparable** | **Comparator** |
| --- | --- | --- |
| Package | java.lang | java.util |
| Sorting logic | In the object itself | External to the object |
| Method | compareTo() | compare() |
| Usage | Natural order | Custom sort (multiple fields) |

**9. What are weak references in WeakHashMap?**

**Answer:**

* Keys in WeakHashMap are stored using **weak references**.
* If a key has no strong reference elsewhere, GC can reclaim it and remove the entry from the map.

**10. What is IdentityHashMap?**

**Answer:**

* Uses == (reference equality) instead of equals().
* Stores keys where **identity matters** (e.g., object intern pool, serialization caching).

**11. How is thread safety handled in collections?**

| **Legacy Thread-safe** | **Modern Alternatives** |
| --- | --- |
| Vector, Hashtable | Collections.synchronizedList, ConcurrentHashMap, CopyOnWriteArrayList |

**Follow-up:** Why avoid Vector/Hashtable now?

They're synchronized on every method call – inefficient compared to modern concurrent classes.

**12. What is the role of EnumMap and EnumSet?**

**Answer:**

* EnumMap: specialized Map for enum keys – efficient and compact.
* EnumSet: high-performance set for enum constants – internally uses bit vectors.

**13. How does PriorityQueue work internally?**

**Answer:**

* Based on a **min-heap** (or max-heap using custom comparator).
* Elements are ordered but not sorted (no guarantee of full ordering when iterating).

**14. Can a Set contain duplicate objects? How is duplication detected?**

**Answer:**

* No. Duplication is checked using equals() and hashCode() in HashSet.

**15. What is the difference between Collection, Collections, and Collectors?**

| **Term** | **Description** |
| --- | --- |
| Collection | Base interface for List, Set, Queue |
| Collections | Utility class with static methods |
| Collectors | Utility class for Java Streams collection |

**🔁 Scenario-based Questions**

**16. How do you avoid ConcurrentModificationException in multi-threaded iteration?**

**Answer:**

* Use ConcurrentHashMap or CopyOnWriteArrayList
* Or use iterator’s remove() method instead of modifying the collection directly.

**17. How would you design a caching layer using collections?**

**Answer:**

* Use LinkedHashMap with **accessOrder=true** + override removeEldestEntry() for LRU cache.
* Or use ConcurrentHashMap for multi-threaded caching with TTL.

**18. How do you handle duplicate records in a List?**

**Answer:**

* Use Set for de-duplication.

java

CopyEdit

List<String> list = Arrays.asList("a", "b", "a");

Set<String> deduped = new HashSet<>(list);

* Or use Stream:

java

CopyEdit

list.stream().distinct().collect(Collectors.toList());

**19. How to sort a list of custom objects?**

**Answer:**

java

CopyEdit

Collections.sort(list, Comparator.comparing(Employee::getName));

or using streams:

java

CopyEdit

list.stream().sorted(Comparator.comparing(Employee::getSalary)).collect(Collectors.toList());

**20. Explain how to implement pagination using collections.**

java

CopyEdit

int page = 2;

int pageSize = 10;

List<T> subList = list.stream()

.skip((page - 1) \* pageSize)

.limit(pageSize)

.collect(Collectors.toList());

**🧠 Java Collections Mock Quiz – Senior Level**

**Q1. What is the output of the following code?**

java

CopyEdit

Map<String, String> map = new HashMap<>();

map.put("one", "1");

map.put("one", "uno");

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

**A.** 2  
**B.** 1  
**C.** 0  
**D.** Compilation error

✅ **Correct Answer: B**  
🔍 **Explanation:** HashMap allows key overriding. The second put() replaces the value for key "one".

**Q2. Which collection class maintains insertion order and allows null keys and values?**

**A.** TreeMap  
**B.** HashMap  
**C.** LinkedHashMap  
**D.** Hashtable

✅ **Correct Answer: C**  
🔍 LinkedHashMap maintains insertion order and supports one null key and multiple null values.

**Q3. What will be the output?**

java

CopyEdit

List<String> list = Arrays.asList("A", "B", "C");

list.add("D");

System.out.println(list);

**A.** [A, B, C, D]  
**B.** Compilation error  
**C.** Runtime Exception  
**D.** [A, B, C]

✅ **Correct Answer: C**  
🔍 Arrays.asList() returns a fixed-size list backed by the array. add() throws UnsupportedOperationException.

**Q4. Which one is NOT fail-fast?**

**A.** ArrayList iterator  
**B.** HashMap keySet iterator  
**C.** ConcurrentHashMap entrySet iterator  
**D.** Vector iterator

✅ **Correct Answer: C**  
🔍 ConcurrentHashMap provides fail-safe iterators.

**Q5. What is the time complexity of HashMap.get() in Java 8 when bucket is a tree?**

**A.** O(1)  
**B.** O(log n)  
**C.** O(n)  
**D.** O(n log n)

✅ **Correct Answer: B**  
🔍 Since Java 8, if collisions form a tree (TreeNode), get() becomes O(log n).

**Q6. Which statement is true about CopyOnWriteArrayList?**

**A.** Best for write-heavy applications  
**B.** Thread-safe and ideal for frequent reads  
**C.** Requires external synchronization  
**D.** It is part of legacy collection classes

✅ **Correct Answer: B**  
🔍 It’s optimized for **read-heavy** concurrent operations (copying happens on write).

**Q7. What does the following code print?**

java

CopyEdit

Set<String> set = new HashSet<>();

set.add("A");

set.add("B");

set.add("A");

System.out.println(set.size());

**A.** 1  
**B.** 2  
**C.** 3  
**D.** 0

✅ **Correct Answer: B**  
🔍 Sets don’t allow duplicates; "A" is added only once.

**Q8. What collection should you use for a bounded, blocking queue in Java?**

**A.** ArrayDeque  
**B.** PriorityQueue  
**C.** ArrayBlockingQueue  
**D.** LinkedList

✅ **Correct Answer: C**  
🔍 ArrayBlockingQueue is ideal for bounded, thread-safe producer-consumer scenarios.

**Q9. What is the issue in this code?**

java

CopyEdit

Set<Employee> set = new HashSet<>();

set.add(new Employee("John", 1));

set.add(new Employee("John", 1));

System.out.println(set.size());

**A.** Always prints 2  
**B.** Compilation error  
**C.** May print 1 if equals() is overridden  
**D.** Runtime exception

✅ **Correct Answer: A**  
🔍 If equals() and hashCode() are **not** overridden, each object is treated as different.

**Q10. Which of these collections guarantees natural ordering of its keys?**

**A.** HashMap  
**B.** TreeMap  
**C.** LinkedHashMap  
**D.** ConcurrentHashMap

✅ **Correct Answer: B**  
🔍 TreeMap uses the natural order or a comparator for keys.

**📘 Advanced Scenario-Based Questions**

**Q11. You need to implement an LRU cache. Which collection should you use?**

**A.** HashMap  
**B.** LinkedHashMap  
**C.** TreeMap  
**D.** Hashtable

✅ **Correct Answer: B**  
🔍 LinkedHashMap with accessOrder=true and removeEldestEntry() overridden.

**Q12. Which queue implementation uses a priority heap internally?**

**A.** ArrayDeque  
**B.** LinkedList  
**C.** PriorityQueue  
**D.** BlockingQueue

✅ **Correct Answer: C**  
🔍 PriorityQueue orders elements using natural/comparator-defined priority.

**Q13. Which will throw a NullPointerException if you insert a null key?**

**A.** HashMap  
**B.** ConcurrentHashMap  
**C.** LinkedHashMap  
**D.** WeakHashMap

✅ **Correct Answer: B**  
🔍 ConcurrentHashMap does not allow null keys or null values.

**Q14. Which data structure to choose for thread-safe, FIFO order queue?**

**A.** ArrayDeque  
**B.** BlockingQueue  
**C.** ConcurrentLinkedQueue  
**D.** LinkedList

✅ **Correct Answer: B**  
🔍 Use LinkedBlockingQueue or ArrayBlockingQueue for thread-safe FIFO queue.

**Q15. What does EnumMap require?**

**A.** Enum values as values  
**B.** Enum values as keys  
**C.** Comparable keys  
**D.** Abstract class inheritance

✅ **Correct Answer: B**  
🔍 EnumMap is optimized for enum **keys only**.