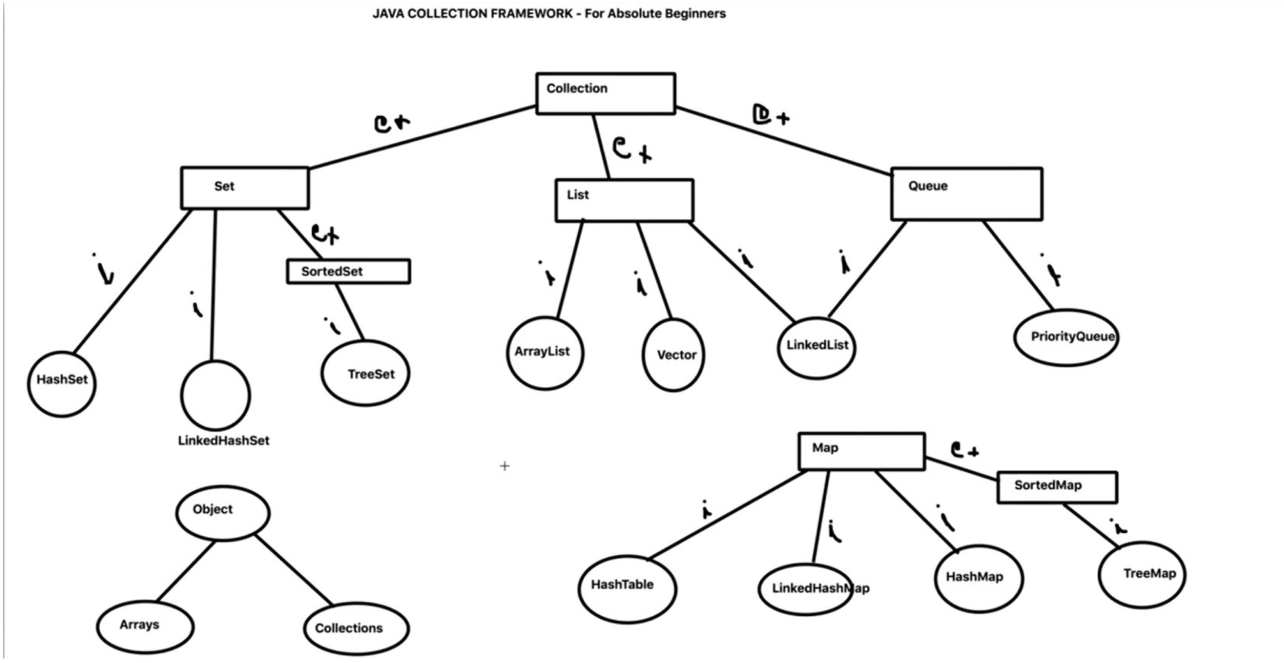
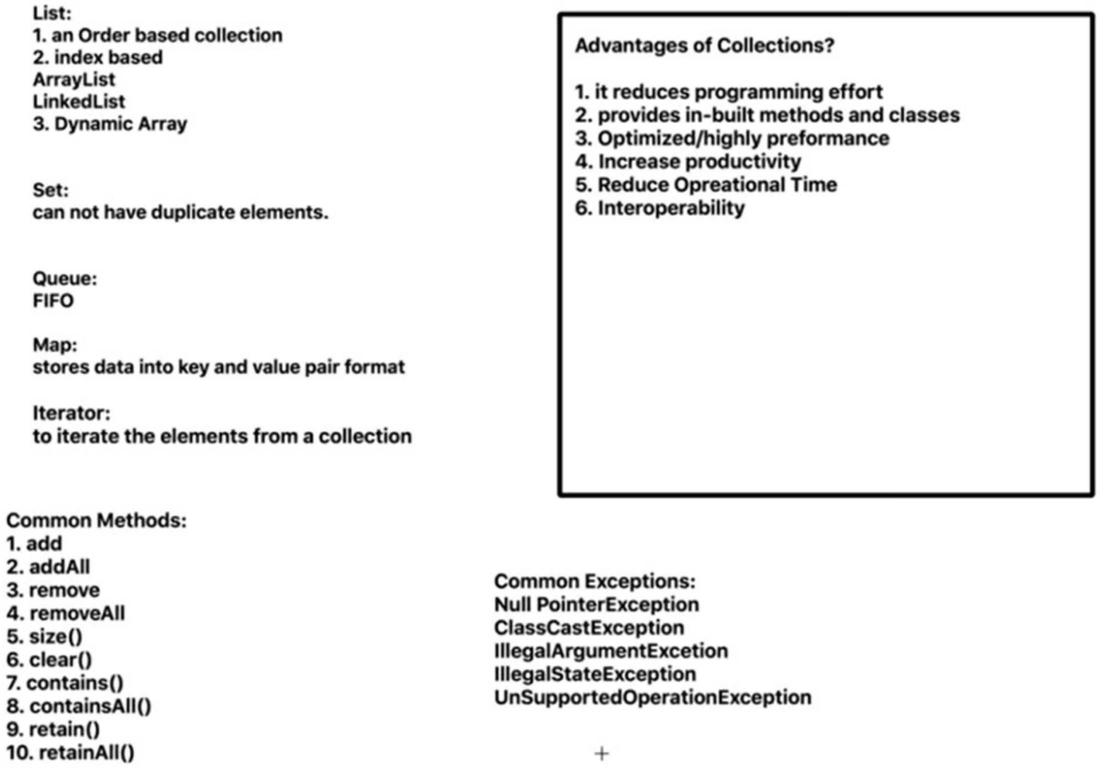


,sdj



List:
1. an Order based collection
2. index based
ArrayList
LinkedList
3. Dynamic Array

Set:
can not have duplicate elements.

Queue:
FIFO

Map:
stores data into key and value pair format

Iterator:
to iterate the elements from a collection

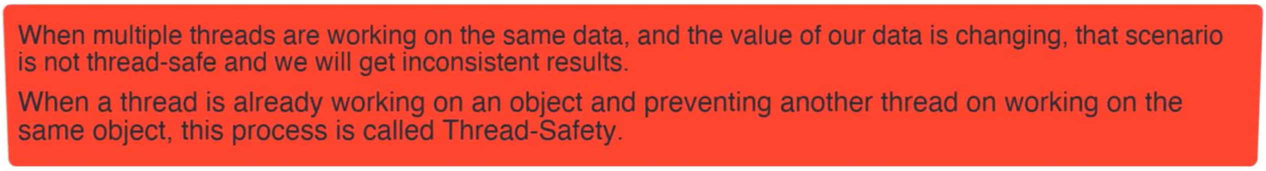
Advantages of Collections?

1. it reduces programming effort
2. provides in-built methods and classes
3. Optimized/highly preformance
4. Increase productivity
5. Reduce Opreational Time
6. Interoperability

Common Methods:
1. add
2. addAll
3. remove
4. removeAll
5. size()
6. clear()
7. contains()
8. containsAll()
9. retain()
10. retainAll()

Common Exceptions:
Null PointerException
ClassCastException
IllegalArgumentExcetion
IllegalStateException
UnSupportedOperationException

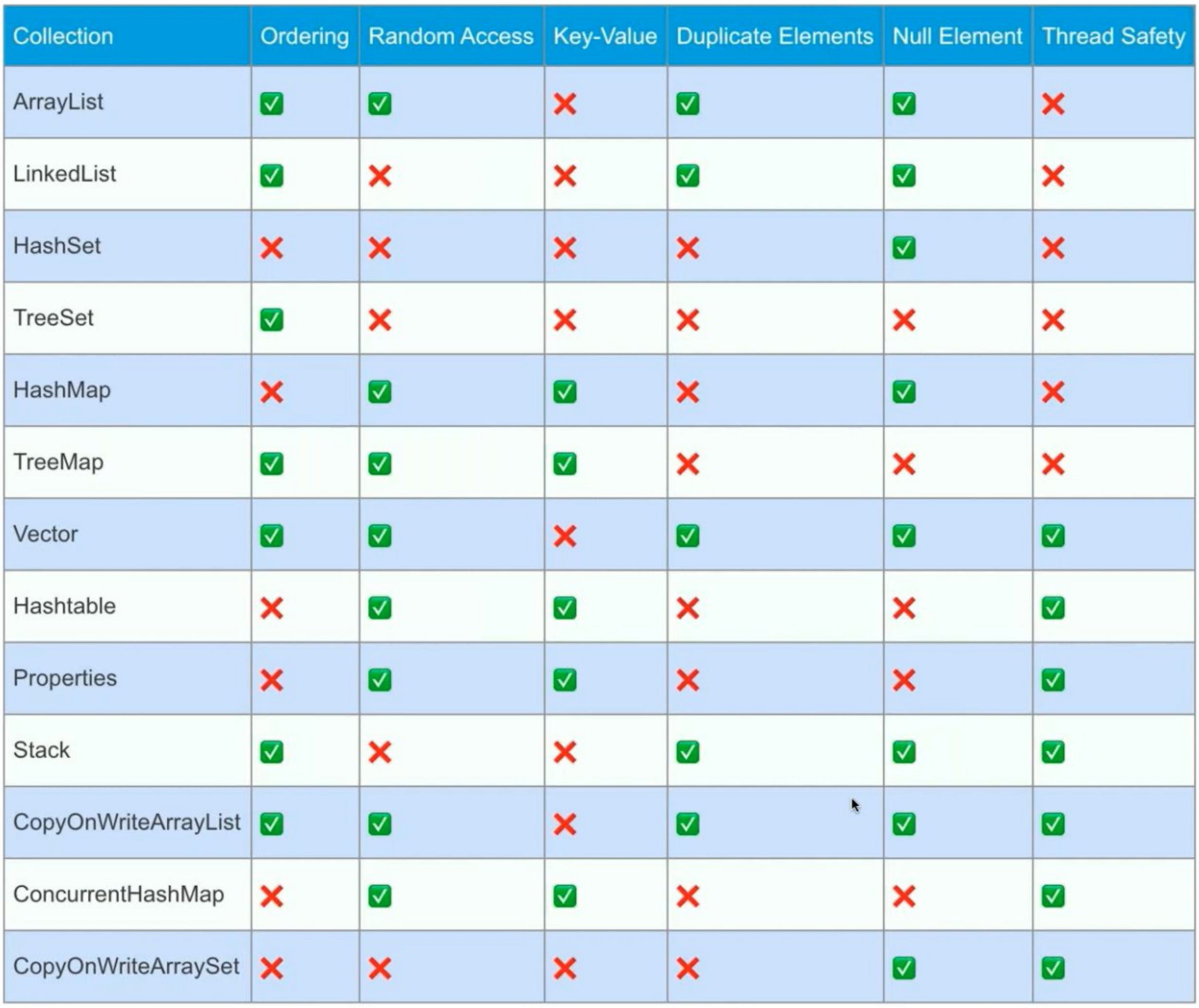
d;lk
rfs



When multiple threads are working on the same data, and the value of our data is changing, that scenario is not thread-safe and we will get inconsistent results.

When a thread is already working on an object and preventing another thread on working on the same object, this process is called Thread-Safety.

jgh



Collection	Ordering	Random Access	Key-Value	Duplicate Elements	Null Element	Thread Safety
ArrayList	✓	✓	✗	✓	✓	✗
LinkedList	✓	✗	✗	✓	✓	✗
HashSet	✗	✗	✗	✗	✓	✗
TreeSet	✓	✗	✗	✗	✗	✗
HashMap	✗	✓	✓	✗	✓	✗
TreeMap	✓	✓	✓	✗	✗	✗
Vector	✓	✓	✗	✓	✓	✓
Hashtable	✗	✓	✓	✗	✗	✓
Properties	✗	✓	✓	✗	✗	✓
Stack	✓	✗	✗	✓	✓	✓
CopyOnWriteArrayList	✓	✓	✗	✓	✓	✓
ConcurrentHashMap	✗	✓	✓	✗	✗	✓
CopyOnWriteArraySet	✗	✗	✗	✗	✓	✓

jhk

1. ArrayList(mummy)

✔ Advantages

- Fast random access (index-based access is O(1))
- Maintains insertion order
- Allows duplicate elements
- Very commonly used and simple

✖ Disadvantages

- **Slow insertion/deletion in the middle (shifting elements)**
- **Not thread-safe**

- **Expansion causes array copy overhead**

⭐ Real-world examples

1. **Shopping cart items (access by index, frequent reads)**
2. **Storing student names in order of admission**

✅ 2. LinkedList

✔ Advantages

- Fast insertion/deletion at beginning or middle
- Maintains insertion order
- Can be used as a queue or stack

✖ Disadvantages

- Slow random access (O(n))
- More memory due to storing previous/next references
- Not thread-safe

⭐ Real-world examples

1. **Browser history navigation (next → previous)**
2. **Customer service queue**

✅ 3. HashSet

✔ Advantages

- Very fast operations (add, remove, search = O(1))
- Automatically removes duplicates
- Good for membership checks

✖ Disadvantages

- No ordering
- Allows only one null
- Not thread-safe

⭐ Real-world examples

1. **Unique usernames in an application**
2. **Tracking visited URLs to avoid processing duplicates**

✅ 4. TreeSet

✔ Advantages

- Stores elements in sorted order
- No duplicates
- Useful for sorted data retrieval

✖ Disadvantages

- Slower than HashSet (operations O(log n))
- No null values allowed
- Not thread-safe

⭐ Real-world examples

1. **Leaderboard scores sorted automatically**
2. **Sorted list of employee IDs**

✅ 5. HashMap

✔ Advantages

- Very fast (O(1) average) key-value retrieval
- Allows one null key and multiple null values
- Flexible and widely used

✖ Disadvantages

- No ordering
- Not thread-safe
- Poor handling in concurrent environment without sync

⭐ Real-world examples

1. **Storing employeeId → employeeDetails**
2. **Caching configurations (key → value)**

✅ 6. TreeMap

✔ Advantages

- Maintains sorted keys
- Good for range queries (headMap, tailMap)

✖ Disadvantages

- Slower than HashMap (O(log n))
- No null keys allowed
- Not thread-safe

⭑ Real-world examples

1. **Dictionary word → meaning (words sorted alphabetically)**
2. **Stock price history sorted by date**

✅ 7. Vector

✔ Advantages

- **Thread-safe (synchronized methods)**
- **Maintains order**
- **Supports random access**

✖ Disadvantages

- Slower due to synchronization
- Legacy class (not recommended)

⭑ Real-world examples

1. **Legacy enterprise apps where Vector was used earlier**
2. **Multi-threaded environment requiring dynamic arrays (old code)**

✅ 8. Hashtable

✔ Advantages

- Thread-safe
- No null keys/values
- Good for legacy concurrent applications

✖ Disadvantages

- Slow because fully synchronized
- Legacy class
- No null support

⭑ Real-world examples

1. **Old banking applications using Hashtable for thread safety**
2. **Storing configuration parameters in old systems**

✅ 9. Properties

✔ Advantages

- Stores configuration in key-value (string → string)
- Can read from .properties files easily
- Easy to load with Properties.load()

✖ Disadvantages

- Only String key/value
- Not suitable for large structured data

⭑ Real-world examples

1. **Reading database config (url, username, password)**
2. **Application settings like timeout, locale**

✅ 10. Stack

✔ Advantages

- LIFO (Last-In-First-Out) structure
- Easy push/pop

✖ Disadvantages

- Synchronized → slow
- Legacy (use Deque instead)

⭑ Real-world examples

1. **Undo/Redo operations in editors**
2. **Backtracking algorithms (DFS)**

✅ 11. CopyOnWriteArrayList

✔ Advantages

- Thread-safe without synchronization blocking

- Best for read-heavy concurrent applications
- No ConcurrentModificationException

✖ Disadvantages

- Very costly on write (copy entire list)
- Not for frequent updates

⭑ Real-world examples

1. **Displaying contacts list in WhatsApp (more reads, fewer edits)**
2. **Caching read-only configurations**

✅ 12. ConcurrentHashMap

✔ Advantages

- Highly efficient thread-safe map
- No full-locking (segment-level locking)
- Very fast in multi-threaded apps

✖ Disadvantages

- No null keys/values
- Slightly more complicated internal structure

⭑ Real-world examples

1. **Storing user sessions in high-traffic applications**
2. **Real-time dashboards requiring concurrent updates**

✅ 13. CopyOnWriteArraySet

✔ Advantages

- Thread-safe
- No duplicates
- Best for read-mostly operations

✖ Disadvantages

- Slow writes
- Uses more memory

⭑ Real-world examples

1. **Unique logged-in user tracking in multi-threaded apps**
2. **Subscription lists (listeners) where reads >> writes**
3. **,khjg**

LinkedHashSet

✔ Advantages

- Maintains insertion order
- Faster than TreeSet
- No duplicates allowed

✘ Disadvantages

- Slightly slower than HashSet
- Uses more memory (because it keeps a linked list)

磋 Real World Example

- Maintaining unique visitors list in the order they visited
  e.g., ["Nisha", "Rahul", "Sneha"]

jk

7. PriorityQueue

✔ Advantages

- Always returns highest or lowest priority element first
- Good for algorithms like Dijkstra, CPU scheduling

✘ Disadvantages

- No random access
- Does not allow null
- Not sorted list — only head has priority

磋 Real World Example

- **Hospital emergency room queue (critical patients first)**
- **CPU task scheduling (high-priority tasks first)**

kh

9. LinkedHashMap

✔ Advantages

- Maintains insertion order
- Faster iteration than HashMap
- Good for building LRU cache

✘ Disadvantages

- Slower than HashMap
- More memory usage (doubly linked list)

礎 Real World Example

- Building LRU Cache
- Storing form data in order fields were entered

lkj

12. Arrays (Utility Class)

✔ Advantages

- Fast and simple
- Good performance
- Fixed-size, memory efficient

✘ Disadvantages

- Cannot grow dynamically
- No built-in functions like sorted, insert at specific position

礎 Real World Example

- Fixed size data storage (like 12 months of a year)

⭑ 13. Collections Utility Class

✔ Advantages

- Provides helper methods
  → sort(), reverse(), shuffle(), min(), max()

✘ Disadvantages

- Works only on Collections, not Maps
- Static methods cannot be overridden

磋 Real World Example

- Sorting list of student marks
- Finding maximum salary from a list

Q1. What is the Java Collections Framework?

Answer:
It is a unified architecture to store and manipulate groups of objects. It includes interfaces (List, Set, Map, Queue), their implementations, and algorithms.

Q2. Difference between ArrayList and LinkedList

Feature	ArrayList	LinkedList
Data structure	Dynamic array	Doubly linked list
Access	Fast (O(1))	Slow (O(n))
Insert/Delete	Slow	Fast
Memory	Less	More

FIS use case:
Transaction logs → ArrayList (read-heavy)

Q3. Difference between HashMap and Hashtable

Feature	HashMap	Hashtable
Thread-safe	✕ No	✓ Yes
Null key/value	Allowed	Not allowed
Performance	Faster	Slower
Legacy	No	Yes

頴 FIS prefers ConcurrentHashMap instead.

Q4. Why ConcurrentHashMap is better than HashMap in multithreading?

Answer:

- Segment-level locking (Java 7) / bucket-level locking (Java 8+)
- Allows multiple threads to read/write concurrently
- No ConcurrentModificationException

FIS relevance: High-volume transaction systems.

Key Reasons:

1. Thread-safe without locking the entire map
2. Better performance than synchronized collections

3. No ConcurrentModificationException
4. Allows concurrent reads and writes
5. Atomic operations like putIfAbsent()

Q5. Difference between List, Set, and Map

Interface	Allows duplicates	Ordered
List	Yes	Yes
Set	No	No
Map	Keys unique	No

INTERMEDIATE → ADVANCED (FIS-Style)

Q6. How does HashMap work internally?

Answer:

- Uses hashing
- Stores entries in buckets
- Uses LinkedList → Red-Black Tree when collisions exceed threshold (Java 8+)

Q7. What is load factor in HashMap?

Answer:
Load factor = 0.75 (default)
Controls resize threshold for performance vs memory tradeoff.

Q8. Can we use a custom object as a key in HashMap?

Answer:
Yes, but must override equals() and hashCode() correctly.

Why important at FIS?
Incorrect hashing → duplicate financial records.

Q9. Difference between fail-fast and fail-safe iterators

Fail-Fast	Fail-Safe
Throws exception	No exception
Works on original collection	Works on copy
Example: ArrayList	Example: ConcurrentHashMap

Q10. What is Collections.synchronizedMap() vs ConcurrentHashMap?

Answer:

- synchronizedMap() → locks entire map
- ConcurrentHashMap → partial locking

顊 **ConcurrentHashMap is preferred in enterprise apps**

REAL-WORLD / SCENARIO QUESTIONS (FIS FAVORITES)

Q11. Which collection will you use for transaction de-duplication?

Answer:
Use HashSet or ConcurrentHashMap
Ensures **uniqueness of transaction IDs**

Q12. How do you sort a list of transactions by amount?

```
transactions.sort(Comparator.comparing(Transaction::getAmount));
```

Q13. How do you make a collection thread-safe?

Answer:

- Collections.synchronizedList()
- CopyOnWriteArrayList
- ConcurrentHashMap

Q14. Difference between Comparable and Comparator

Comparable	Comparator
Natural ordering	Custom ordering
compareTo()	compare()
One logic	Multiple logics

Q15. What is CopyOnWriteArrayList?

Answer:

- Thread-safe
- Creates a new copy on write
- Best for read-heavy systems

FIS Example:
Reference data, configuration caches

ADVANCED (Bonus – Stand Out)

Q16. How does Java 8 improve collections?

- Stream API
- Lambda expressions
- Default methods
- `forEach()`

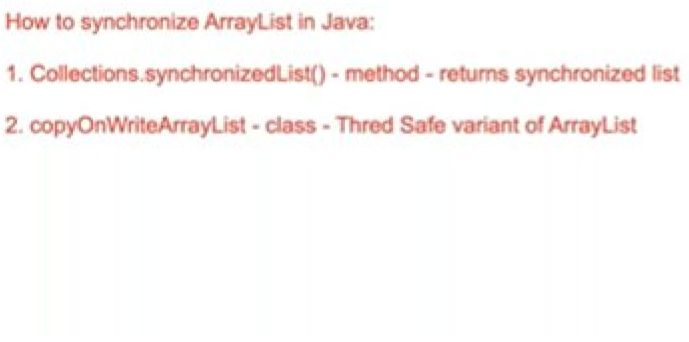
Q17. How do you avoid memory leaks in collections?

Answer:

- Remove unused references
- Use WeakHashMap
- Avoid static collections

Q18. Which collection is best for high-frequency trading systems?

Answer:
ConcurrentHashMap, ArrayBlockingQueue, LinkedBlockingQueue



How to synchronize ArrayList in Java:

1. Collections.synchronizedList() - method - returns synchronized list

2. copyOnWriteArrayList - class - Thred Safe variant of ArrayList