**Basic Questions:**

**What is the Java Collection Framework?**

1. A unified architecture for storing, manipulating, and processing collections of data.
2. Provides **interfaces, classes, and algorithms** for data structures like lists, sets, maps, and queues.
3. Core Interfaces: Collection (Root interface for data structures storing elements), Map (Interface for key-value pairs).

**What are the main advantages of using the Collection Framework?**

1. It provides standard utility methods like add(), remove(), size() etc.
2. Contains ready to use data structures like Arraylist, HashSet and HashMap that saves time.
3. **Dynamic sizing -** Collections can grow or shrink as needed, unlike arrays.
4. Handles big dataset.
5. **Provides thread safety and Synchronization** to multi-threaded programs.

**What is the difference between a Collection and a Collections class?**

**Collection :**

1. **Collection** is an **interface in JCF** — the foundation of all data structures in Java (list, set, queues).
2. Implemented by interface like List, Set and Queues
3. Contains abstract methods like add(), remove() size() etc

**Collections :**

1. **Collections** is a **utility class** — provides methods to manipulate data structures.
2. Cannot be extended – it’s a final class
3. Contains static methods like sort(), reverse(), shuffle() etc.

**Explain the core interfaces of the Java Collection Framework.**

The JCF has two main root interfaces:

1. Collection (for storing individual elements)
2. Map (for storing key-value pairs)

**What are the differences between List, Set, and Map interfaces?**

List:

1. Ordered collection (Maintains insertion order of elements).
2. Allows duplicates and multiple null values.
3. Fast random access (ArrayList) & slower for insert/remove (LinkedList).
4. Main classes are ArrayList, LinkedList, vector, Stack.
5. No automatic sorting.

Set:

1. Unordered collection of unique elements (Doesn’t guaranteed order except LinkedHashSet).
2. Duplicates not allowed & allows one null value.
3. Fast lookups (HashSet), slower if using sorted sets (TreeSet).
4. Main classes are HashSet, LinkedHashSet, TreeSet.
5. TreeSet sorts elements by natural order or custom comparator.

Map:

1. Collection of key-value pairs. No order (except LinkedHashMap or sorted in TreeMap).
2. Keys must be unique, but values can duplicate.
3. Fast lookups by key (HashMap), slower in TreeMap.
4. Main classes are HashMap, LinkedHashMap, TreeMap, Hashtable.
5. TreeMap sorts keys by natural order or custom comparator.

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

ArrayList :

|  |
| --- |
|  |

1. Uses a **dynamic array** to store elements.
2. Faster in terms of accessing the elements using get() as it is stored in **contiguous** memory location.
3. Insertion or deleting elements in the middle of the list is slower due to shifting elements.
4. writing operating is time consuming coz when ArrayListis full it will shift all data to another ArrayListof big size this will consume more time
5. **Default size is 10** (expands by 50% when full).
6. **Fail-fast**

LinkedList:

1. Uses a **doubly linked list** structure.
2. Slower In terms of random access as it traverses from the start till the elements to be searched.
3. Faster for insert and remove operation as only the previous and next node’s address needs to be changed.

**Vector :**

1. Vector is a **legacy class** In Java that implements the List interface and is part of the java.util package.
2. Allows duplicate and null values.
3. Maintains insertion order.
4. Vector is **synchronized** and hence **thread-safe** by default – but this makes it **slower** **than ArrayList** in single-threaded scenarios.
5. Automatically resizes when elements are added, default size is 10 is not specified during creation.
6. Increase its size by 100%
7. **Fail-safe (but may throw ConcurrentModificationException)**

**Difference between ArrayList and Array**

**Array:**

1. It is fixed in size.
2. There are no built-in methods.
3. Can only store primitives and objects.
4. Faster than ArrayList.

**ArrayList:**

1. It is dynamic in nature.
2. Contains built-in methods.
3. Stores only objects.
4. Slightly slower than Array.

**How is a HashSet different from a TreeSet?**

HashSet:

1. Backed by a **HashMap**.
2. **Unordered** — elements are not sorted.
3. Allows **one null value**.
4. Uses **equals()** and **hashCode()** for comparison.
5. Best for **Fast lookups** and when ordering doesn’t matter.

TreeSet :

|  |
| --- |
|  |

1. Backed by a **TreeMap** (Red-Black tree)
2. **Sorted** in **natural order** (ascending) or using a custom comparator.
3. **Allows only homogeneous type of data (throws ClassCastException)**
4. Does **not allow null values** (throws NullPointerException).
5. Uses **compareTo()** (or comparator) for ordering and comparison.
6. Best for **Sorted data** — when you need to keep elements in order.

**What is the difference between HashMap and Hashtable?**

HashMap:

1. **Unordered** – It does not maintain insertion order; the order of retrieval may differ.
2. **Stores key-value pairs using hashing** – internally uses an array + hashCode to distribute keys.
3. **Not thread-safe**, hence **faster**; can be made thread-safe using Collections.synchronizedMap().
4. **Allows one null key** and **multiple null values**.

HashTable:

1. **Legacy class** – Thread-safe alternative to HashMap, introduced in Java 1.0.
2. **Unordered** – Does not maintain insertion order.
3. **Thread-safe and synchronized** – All methods are synchronized; only one thread can access it at a time.
4. **Does not allow null keys or null values** – Adding them will throw a NullPointerException.

**Can a List contain duplicate elements?**

Yes, List is an implementation of Collection Framework that allows multiple duplicate elements. This is because **List** is an **ordered collection** — it maintains the **insertion order** and allows elements to be accessed by their index.

**Intermediate Questions:**

**How does HashMap work internally?**

1. It internally uses **array of nodes(bucket)**, each node contains **key, value, hash and address of next node**.
2. When we insert the data inside HashMap by put(), first it calculates hashCode and then index value by dividing hashcode to size of array and remainder will be considered as index.
3. If bucket is empty then add entry directly, if there’s collision then check if keys are equal and if keys are equal then value gets updated else add the node to the linked list(**Singly linkedlist**) or tree (in java 8)
4. Before java 8 HashMap used **linked list** to resolve collision, after that it uses **red-Black tree** if bucket contains more than 8 nodes
5. When we use get() it calculates bucket index & if there a single value it returns directly if there’s multiple values then traverse and check equals() and if a tree then perform tree search

**What is the load factor in HashMap?**

1. Default load factor is **0.75 (75%).**
2. When the number of key-value pairs exceeds **75% of the current capacity**, the HashMap resizes by **doubling the number of buckets**.

**What are fail-fast and fail-safe iterators?**

**Fail-fast :**

1. These iterators throw a **ConcurrentModificationException.** If the collection is modified structurally during iteration.
2. They check for modification counts – if the count changes unexpectedly, they fail immediately.
3. Collections using fail-fast iterators: Arraylist, HashMap, LinkedList, HashSet.
4. Not thread safe – may fail if modified during iteration.
5. Iterates over the original collection.
6. Faster but risky in concurrent environments.

**Fail-safe:**

1. These iterators don’t throw exceptions if the collection is modified during iteration because they operate on a copy of the collection.
2. They iterate over a cloned structure, so changes to the original collection don’t affect iteration.
3. Collections using fail-safe iterators : ConcurrentHashMap, CopyOnWriteArrayList.
4. Designed for concurrent access.
5. Iterates over copy.
6. Slower due to copying but safe for concurrency.

**What is the difference between Comparable and Comparator interfaces?**

**Comparable :**

1. **Comparable** is an interface that needs to be implemented .
2. It contains compareTo() method
3. Needs to override inside the current class and it is modified according to the requirements
4. Only **one sorting** logic can be done
5. Can compare **primitives and objects**

**Comparator** :

1. Comparator is an interface that needs to be implemented. It gets more preference than comparable
2. It contains compare() method with two arguments
3. It is written outside the existing class.
4. We can create n number of classes to override comareTo() according to logic, **n number of sorting** logic can be provided
5. It compares only **objects**

**How does a TreeMap ensure the elements are in sorted order?**

TreeMap internally uses Comparable & Comparator to sort the keys.

Keys are placed based on natural ordering (Comparable) or custom rules (Comparator)

**What are the differences between synchronized and concurrent collections?**

**Synchronized :**

1. Locks entire collection.
2. Requires manual synchronization.
3. Slower due to full locking.
4. Vector, HashTable, synchronized wrappers.
5. Throws ConcurrentModificationException.

**Concurrent Collection:**

1. Fine grained or non-blocking mechanism.
2. Thread safe – fail-safe iterators.
3. Faster due to partial or no locking.
4. ConcurrentHashMap, CopyOnWriteArrayList.
5. No exception during iteration.

**What are the roles of equals() and hashCode() methods in Java collections?**

**Equals()**

1. It is a method of Object class
2. Checks whether two objects are **logically same**.
3. By default, it checks the reference & if we override then it checks for the content.
4. Used in HashSet & HashMap

**hashCode()**

1. it is a method inside the Object class
2. it returns the integer hash code for an object provided buy JVM. **Used to determine the bucket** **index** where the object will be stored.
3. By default, it returns the unique hash code & if we override then the equal objects will have same hash code.

**What is the difference between Arrays and Collections in Java?**

**Arrays:**

1. Array is fixed in size; we cannot change the size after its declaration.
2. It is homogeneous in nature. Can store primitives and objects
3. Stored in contiguous memory location.
4. have limited built in methods.
5. **Fast for accessing the elements**

**Collections:**

1. It is dynamic in size.
2. It is heterogeneous in nature. It stores only objects.
3. have many built in methods.
4. **Slower in accessing elements due to extra functions and dynamic resizing.**

**How does the LinkedHashMap maintain the order of elements?**

1. LinkHashMap uses doubly-linked list with the standard hash table.
2. nodes store the address of previous and next element that links to its predecessor and successor.
3. These forms linked list of entries, that maintain the insertion order

**What is the difference between IdentityHashMap and HashMap?**

**IdentityHashMap :**

1. Uses reference equality (==) to compare keys.
2. Keys are equal only if they reference the same object.
3. Allows multiple null keys and null values.
4. Used when you need to track object identity, working with object graphs, referenced based key comparison.

**HashMap :**

1. Uses the key equals() & hashCode() method.
2. Keys are logically equal if equals() return true.
3. Allows single null key and multiple null values.
4. Used when we want key equality based on logical comparison

**Iterable vs Iterator**

**Iterable : (Interface)**

* Introduced in Java 5
* It represents that a collection can be iterated using an Iterator.
* It has only iterator().

**Iterator : (Interface)**

* Provides the methods to iterate/traverse the collection.
* hasNext(), hasPrevious(), next(), remove().

**Where we will store Null Key HashMap?**

1. When you insert an entry with a null key, it bypasses the normal hashcode-based logic.
2. Instead of calling null.hashCode() (which would throw a NullPointerException), the HashMap internally checks:

**if (key == null) {**

**// store at bucket index 0**

**}**

1. **So, the null key is always stored at bucket index 0 in the internal array (called the table).**

**How to sort hashmap?**

Technically, no — a HashMap doesn't maintain order.  
But you can extract its entries, sort them, and store them in something order-preserving, like:

* LinkedHashMap (preserves insertion order)
* TreeMap (sorted by key)
* List<Map.Entry<K, V>> (for full custom sorting)

**If you're using a Set<Employee> — will it allow duplicate employees?**

Set doesn’t allow duplicates, but whether two Employee objects are considered duplicates depends on how equals() and hashCode() are implemented. If not overridden, it allows duplicates by value but not by reference.

**What is a Hashcode Collision?**

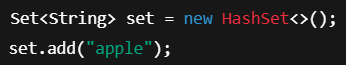
A hashcode collision happens when two different objects produce the same hashcode value.

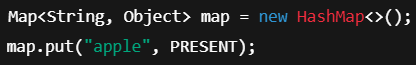
**Why collision occur ?**

since Java’s hashCode() method returns a 32-bit int, the number of unique hash codes is limited. That means different objects can have the same hash code — a situation called a hash collision. This is perfectly valid as long as the contract between equals() and hashCode() is maintained, and Java's collections like HashMap are designed to handle such collisions internally.

**if HashSet is backed by HashMap then how HashMap store HashSet objects?**

each element of the set stored as a **key** in the map.





PRESENT is a **dummy constant object** (typically private static final Object PRESENT = new Object();) used as the **value** in the map.

The actual elements of the HashSet ("apple", "banana", etc.) are stored as **keys** in the HashMap.