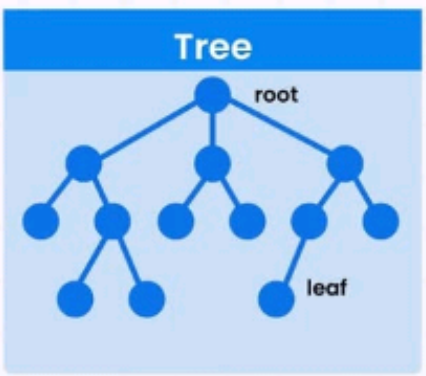
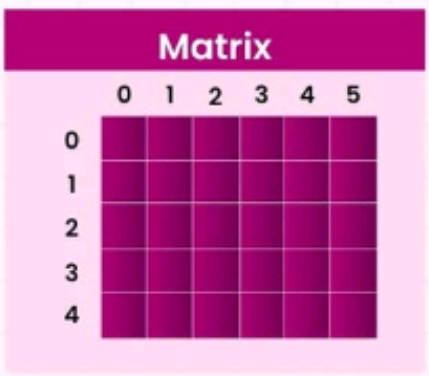
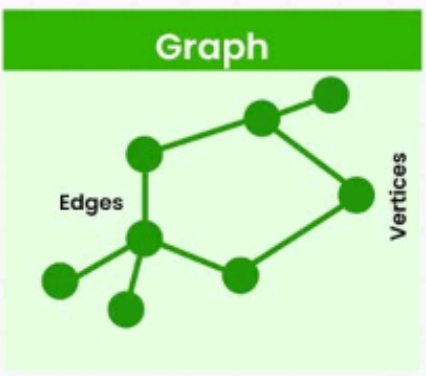
Java Collections Framework





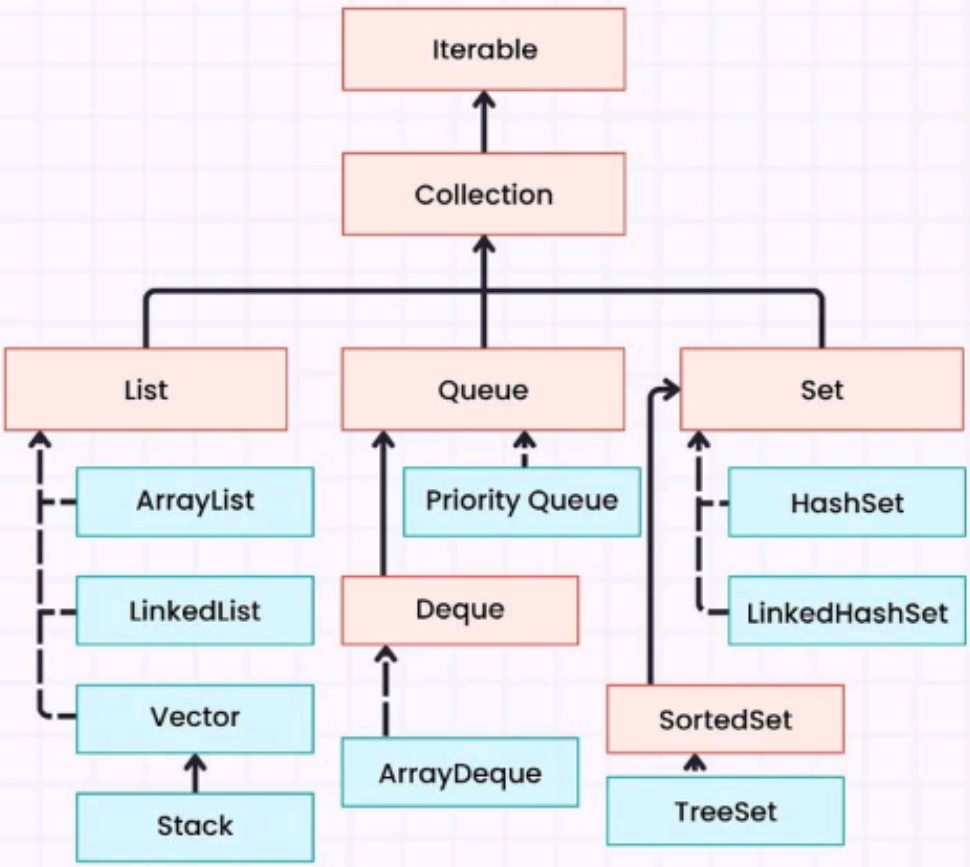


SWIPE

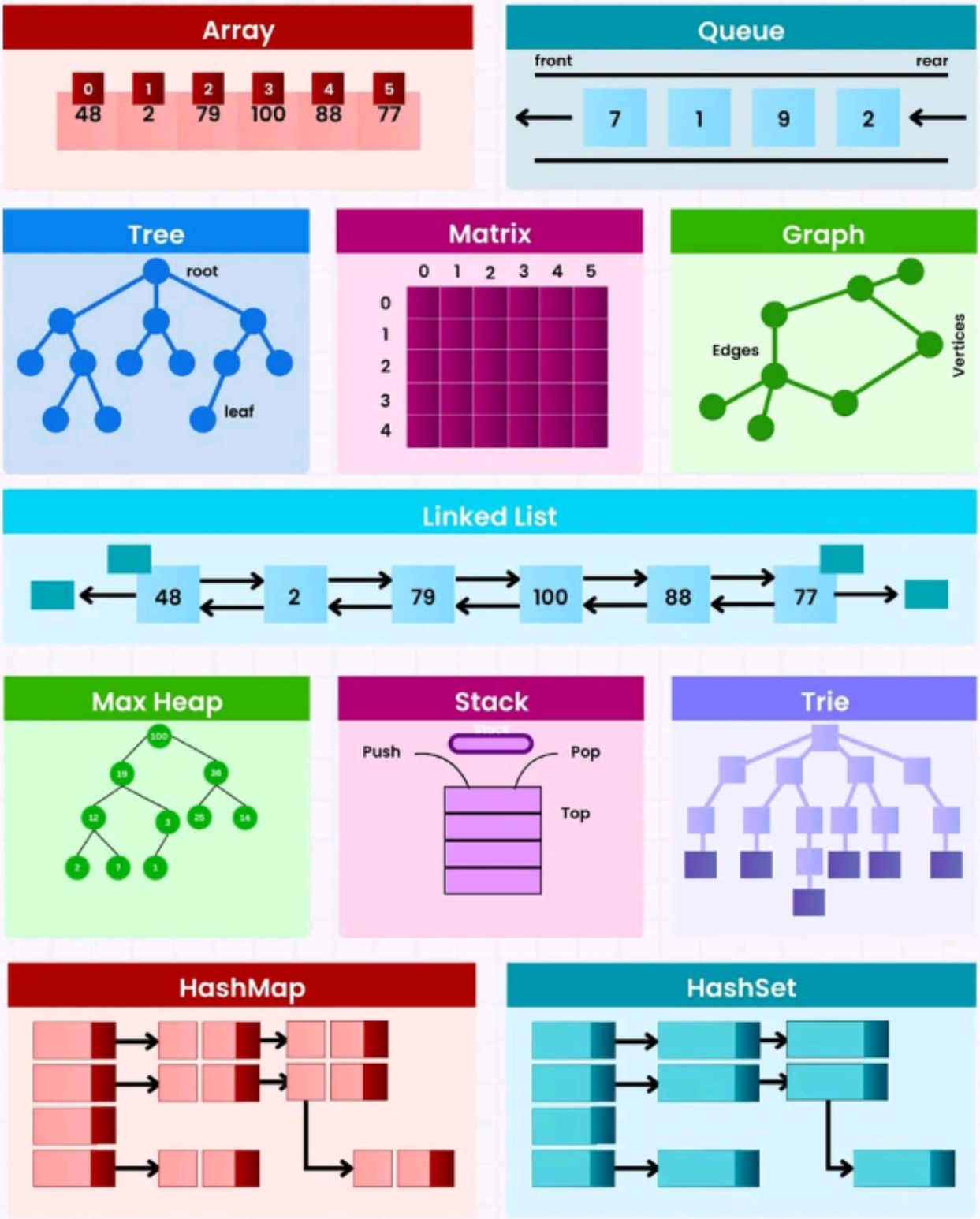
Java Collections Framework

Introduction

- The Java Collections Framework (JCF) is a fundamental part of the Java programming language, providing a set of classes and interfaces for storing and manipulating groups of data.
- The framework offers a range of collection types, including List, Set, and Map, each tailored for different use cases.
- It also provides mechanisms for sorting and comparing data, making it essential for efficiently handling large amounts of information in applications.
- In this chapter, we will cover the core collection types, compare HashMap and HashTable, and explore the differences between Comparable and Comparator interfaces.



- The threemost commonly used collection types in Java are List, Set, and Map.
- Each serves a distinct purpose in managing and storing elements.

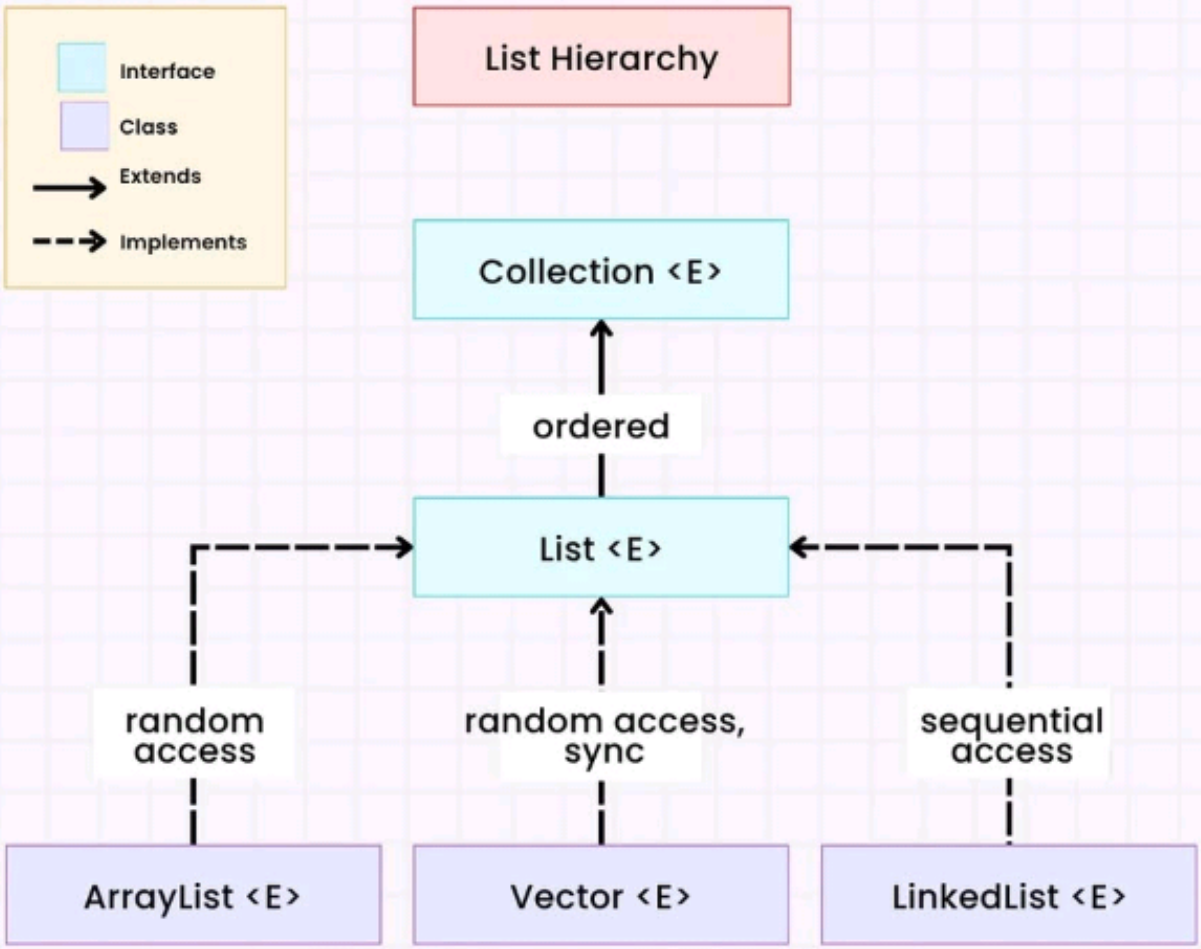


💬 Comment "Java" and get the complete PDF in your DMs! 📩

List:

- **Definition:** A List is an ordered collection that allows duplicates.
- Elements can be accessed by their position (index).
- Common Implementations: ArrayList, LinkedList.
- **Use Cases:**
  - When you need to maintain the order of insertion.
  - When duplicate elements are allowed.

```java
List<String> names = new ArrayList<>();
names.add("John");
names.add("John");  // Duplicate allowed
names.add("Doe");
System.out.println(names);  // Output: [John, John, Doe]
```



Set:

- **Definition:** A Set is a collection that does not allow duplicate elements and does not guarantee order.
- Common Implementations: HashSet, LinkedHashSet, TreeSet.
- **Use Cases:**
  - When you want to avoid duplicates.
  - When the order of elements is not important (for HashSet), or when you need sorted elements (for TreeSet).

```java
Set<String> namesSet = new HashSet<>();
namesSet.add("John");
namesSet.add("John");  // Duplicate ignored
namesSet.add("Doe");
System.out.println(namesSet);  // Output: [John, Doe]
```

Map:

- **Definition:** A Map is a collection that maps keys to values, where each key is unique but values can be duplicated.
- Common Implementations: HashMap, TreeMap, LinkedHashMap.
- **Use Cases:**
  - When you need to associate unique keys with values.
  - When fast lookups by key are essential.

```java
Map<String, Integer> ageMap = new HashMap<>();
ageMap.put("John", 25);
ageMap.put("Doe", 30);
System.out.println(ageMap);  // Output: {John=25, Doe=30}
```

Real-Life Example

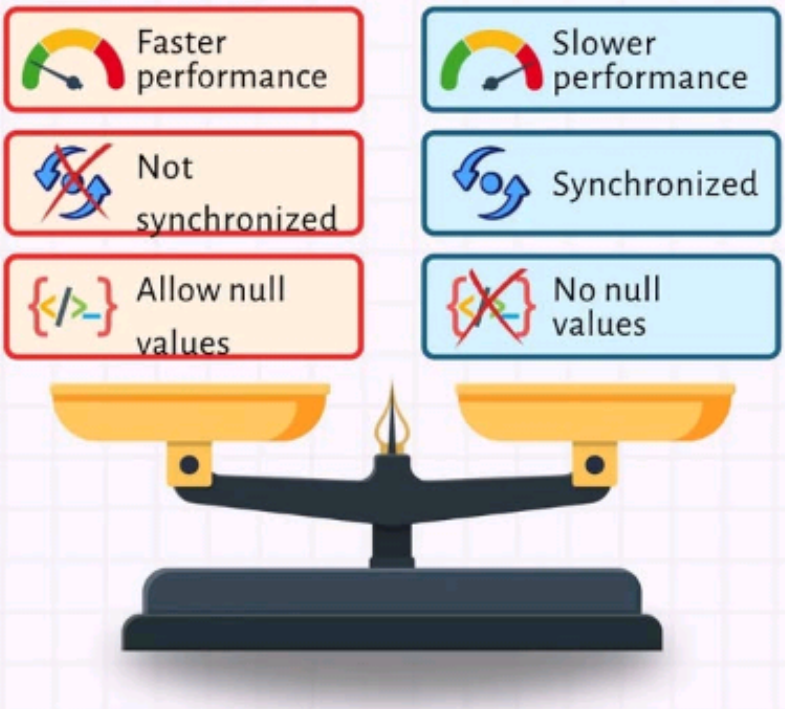
Think of a List as a queue of people waiting in line where the order matters, a Set as a guest list where you want to ensure no duplicates, and a Map as a phone directory where each name (key) is associated with a phone number (value).

Tip

Use a List when order matters, a Set when you need uniqueness, and a Map when you want to associate keys with values.

2. HashMap vs. HashTable

Both HashMap and HashTable are implementations of the Map interface in Java, but there are some significant differences between the two.



Comparing HashMap and HashTable in Java

HashMap:

- Allows null values: HashMap allows one null key and multiple null values.
- Not synchronized: HashMap is not synchronized, meaning it is not thread-safe and should not be used in multi-threaded environments without external synchronization.
- Faster: Due to the lack of synchronization, HashMap generally performs faster in single-threaded environments.

HashTable:

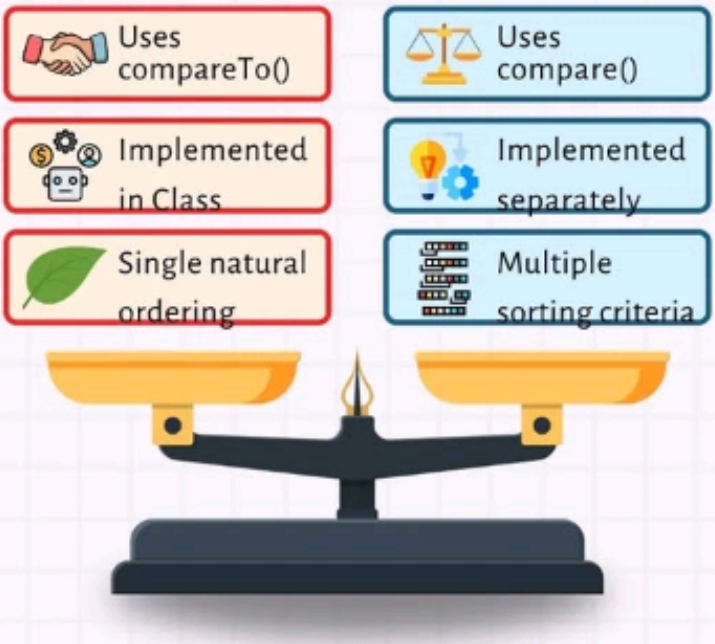
- Does not allow null values: HashTable does not permit null keys or values.
- Synchronized: HashTable is synchronized, meaning it is thread-safe and can be used in multi-threaded environments. However, synchronization makes it slower compared to HashMap.
- Legacy class: HashTable is considered a legacy class and is generally discouraged in favor of HashMap and ConcurrentHashMap.

Real-Life Example

Imagine HashMap as a self-service buffet where people can serve themselves quickly but need to be careful not to collide, while HashTable is like a waiter-served restaurant where you must wait for your turn, ensuring no conflicts but with slower service.

3. Comparable vs. Comparator

Sorting is an essential operation for collections. Java provides two mechanisms for sorting: Comparable and Comparator.



Comparing between Comparable and Comparator for sorting in Java

Comparable:

- **Definition:** The Comparable interface defines a natural ordering of objects. It is implemented directly by the class and allows objects to be compared using the compareTo() method.
- **Use Case**: When you want to define a single natural ordering for a class.

```
class Employee implements Comparable<Employee> {
    String name;
    int age;

    @Override
    public int compareTo(Employee other) {
        return this.age - other.age;  // Compare based on age
    }
}
```

Comparator:

- **Definition:** The Comparator interface allows you to define multiple sorting criteria. It is implemented separately from the class and provides a compare() method for sorting.
- **Use Case:** When you want to define multiple ways to compare objects.

```
class EmployeeAgeComparator implements Comparator<Employee> {
    @Override
    public int compare(Employee e1, Employee e2) {
        return e1.age - e2.age;  // Compare based on age
    }
}

class EmployeeNameComparator implements Comparator<Employee> {
    @Override
    public int compare(Employee e1, Employee e2) {
        return e1.name.compareTo(e2.name);  // Compare based on name
    }
}
```

Real-Life Example

Think of Comparable as a natural rank order, like people's ages (youngest to oldest). Comparator is like different ways to sort—by age, height, or even by name, depending on what you want to focus on.

Tip

Use Comparable for natural ordering within the class and Comparator for custom sorting, where flexibility in sorting logic is required.

Summary

- The Java Collections Framework is vital for efficient data management in Java applications.
- By understanding how to use List, Set, and Map, and knowing when to use Comparable or Comparator for sorting, you can build flexible and high-performing Java programs.
- Whether you're working with simple lists or complex data structures, mastering these concepts is key to becoming a proficient Java developer.

Interview Q & A

1. What is the Java Collections Framework?

- The Java Collections Framework (JCF) is a unified architecture for representing and manipulating collections of objects.
- It provides interfaces (such as List, Set, and Map) and classes (such as ArrayList, HashSet, and HashMap) that implement various collection types.

Quick Notes

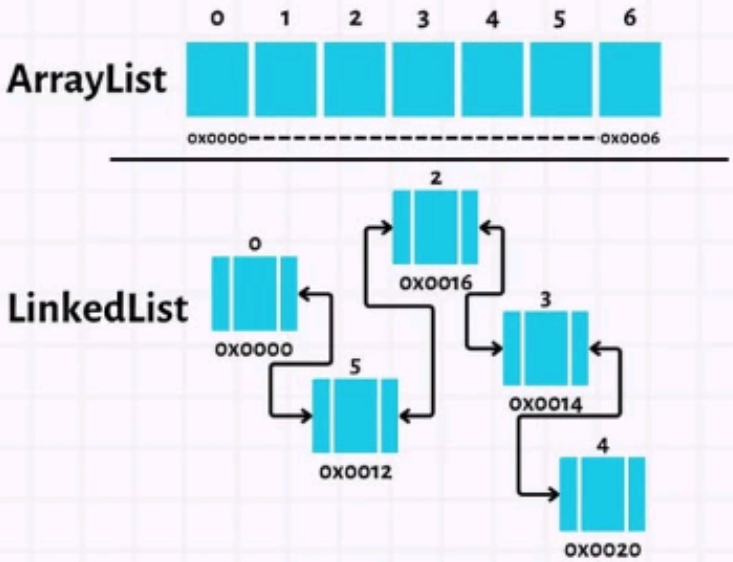
JCF standardizes data structure handling in Java, allowing developers to work with collections in a consistent way.

2. What is the difference between a List, Set, and Map?

- List: An ordered collection that allows duplicates (e.g., ArrayList, LinkedList).
- Set: An unordered collection that does not allow duplicates (e.g., HashSet, TreeSet).
- Map: A collection of key-value pairs where keys are unique (e.g., HashMap, TreeMap).

3. How is an ArrayList different from a LinkedList?

- ArrayList: Backed by an array, better for random access but slower for insertions and deletions.
- LinkedList: Doubly-linked list structure, better for insertions and deletions but slower for random access.



Difference Between ArrayList & LinkedList

4. What is the main advantage of using a HashSet over a List?

- HashSet does not allow duplicates and provides constant-time complexity for add, remove, and contains operations.
- It's ideal when you need a collection of unique elements.

5. How does a TreeSet maintain order?

- TreeSet maintains elements in natural order or a specified comparator order.
- It is implemented using a self-balancing binary search tree (usually a Red-Black Tree).

6. Explain the key differences between a HashMap and a HashTable.

- HashMap: Not synchronized, allows one null key and multiple null values.
- HashTable: Synchronized, does not allow null keys or values, considered legacy.

7. Why would you choose a ConcurrentHashMap over a HashTable?

ConcurrentHashMap is designed for concurrent access, offering better performance than HashTable in a multithreaded environment by using locks at a more granular level.

8. Can a Set contain duplicate elements?

No, a Set cannot contain duplicate elements. Any attempt to add a duplicate to a Set will be ignored.

9. How does the HashMap handle collisions?

- HashMap uses chaining to handle collisions, where each bucket contains a linked list of entries with the same hash code.
- In Java 8, a tree structure replaces the linked list for large buckets to improve performance.

10. When should you use a LinkedHashMap instead of a HashMap?

LinkedHashMap maintains insertion order, making it useful when you need predictable iteration order along with the key-value mapping of a HashMap.

11. What is the difference between Comparable and Comparator?

- Comparable: Interface used to define a natural order within the class itself (compareTo method).
- Comparator: Interface used to define custom orders outside the class (compare method).

12. How would you sort a list of objects by multiple criteria?

Use a Comparator chain to compare objects by multiple fields. Java 8 provides Comparator.thenComparing() for this purpose.

```
Comparator<Employee> comparator = Comparator.comparing(Employee::getLastName)
                                            .thenComparing(Employee::getFirstName);
```

13. What happens if two keys have the same hash code in a HashMap?

If two keys have the same hash code, they are stored in the same bucket. The equals() method is then used to differentiate between keys.

14. Can a HashMap have a null key?

Yes, a HashMap can have one null key and multiple null values.

15. Why is HashTable considered a legacy class?

HashTable is a legacy class because it's synchronized, which is less efficient in modern concurrent applications. ConcurrentHashMap is preferred for thread-safe operations.

16. How does the TreeMap maintain the order of keys?

TreeMap sorts keys in natural order or based on a provided comparator. It's implemented as a Red-Black Tree, ensuring sorted keys.

17. How does a HashSet check for duplicates?

HashSet uses the hashCode() and equals() methods to determine if two objects are identical, thus preventing duplicates.

18. What is the difference between HashSet and LinkedHashSet?

- HashSet: Does not maintain any order.
- LinkedHashSet: Maintains insertion order by linking entries in the order they were added.

19. How would you synchronize a HashMap?

You can create a synchronized version of a HashMap using Collections.synchronizedMap().

```
Map<String, String> map = Collections.synchronizedMap(new HashMap<>());
```

20. What is the default capacity of a HashMap?

The default capacity of a HashMap is 16.

21. Can you explain how the load factor affects HashMap performance?

- The load factor determines when to resize the HashMap.
- A lower load factor reduces space efficiency, while a higher load factor may increase the chances of collisions.

22. Why is it necessary to implement hashCode() and equals() methods when using collections like HashSet?

- hashCode() and equals() methods are used to identify duplicates in collections like HashSet and HashMap.
- Correctly implemented methods ensure reliable storage and retrieval.

23. How do you iterate over the keys of a HashMap?

You can use a for-each loop with keySet() to iterate over keys.

```
for (String key : map.keySet()) {
    System.out.println(key);
}
```

24. Can you explain the concept of a backed collection in Java?

A backed collection is a collection that reflects changes made to another collection.
For instance, subList() in an ArrayList is a backed collection.

25. What is the difference between TreeSet and TreeMap?

- TreeSet: Stores unique elements in sorted order.
- TreeMap: Stores key-value pairs in sorted order of keys.

26. When would you use a WeakHashMap?

Use WeakHashMap for memory-sensitive caches, as entries are removed automatically when keys are no longer referenced.

27. What is the role of Collections.synchronizedMap()?

Collections.synchronizedMap() wraps a regular map, providing a synchronized (thread-safe) version.

28. How would you sort a List using the Comparable interface?

Implement Comparable in the class and use Collections.sort().

```
class Person implements Comparable<Person> {
    String name;
    public int compareTo(Person other) {
        return this.name.compareTo(other.name);
    }
}
Collections.sort(personList);
```

29. How would you sort a List using the Comparator interface?

Implement a custom Comparator or use a lambda function.

```
Collections.sort(personList, Comparator.comparing(Person::getName));
```

30. What is the role of the iterator() method in Java collections?

The iterator() method returns an Iterator to traverse a collection safely and sequentially.

31. What are the benefits of using a LinkedHashSet over a HashSet?

LinkedHashSet maintains insertion order, which is useful when the order of elements matters.

32. How do you merge two maps in Java?

Use putAll() to merge maps, or in Java 8, use merge() for custom merging.

```
map1.putAll(map2);
```

33. How would you reverse the order of elements in a List?

Use Collections.reverse() to reverse the order.

```
Collections.reverse(list);
```

34. What is a NavigableMap, and how does it work?

NavigableMap extends SortedMap and provides navigation methods like lowerKey(), floorKey(), etc., for retrieving keys based on proximity.

35. How does the subList() method work in an ArrayList?

The subList() method returns a portion of the ArrayList between specified indices, and it's a backed collection, so changes reflect in the original list.

36. What is the difference between Collections.sort() and Arrays.sort()?

- Collections.sort(): Sorts elements in a collection like List.
- Arrays.sort(): Sorts elements in an array.

37. Explain how to implement a custom comparator for a class.

Implement the Comparator interface or use a lambda.

```
Comparator<Person> byName = (p1, p2) -> p1.getName().compareTo(p2.getName());
```

38. How does a PriorityQueue order its elements?

A PriorityQueue orders elements according to natural order or a custom comparator, with the highest priority at the head of the queue.

39. What are the key differences between a Queue and a Stack?

- Queue: FIFO (First-In-First-Out).
- Stack: LIFO (Last-In-First-Out).

40. How can you convert a List to a Set in Java?

Use the HashSet constructor to convert a List to a Set.

```
List<String> list = new ArrayList<>(Arrays.asList("A", "B", "C"));
Set<String> set = new HashSet<>(list);
```