JAVA Collection Framework

Common Interview Questions

1. **What is a Collection in Java?**

A collection in Java refers to a framework that allows us to store and manage groups of objects as a single unit. It provides a way to organize, store, and manipulate data efficiently.

With the introduction of JDK 1.2, Java introduced the “Collection Framework,” which provides a standardized way of working with collections.

Collections in Java can handle various tasks like finding items, organizing, adding, changing, or removing them.

2. **What are the main interfaces of the Java Collection Framework?**

The main interfaces are:

- **List:** An ordered collection that allows duplicate elements.
- **Set:** A collection that does not allow duplicate elements.
- **Map:** A collection that maps keys to values, where each key is unique.

3. **What is the difference between Array and Collection in Java?**

Array	**Collection**

Arrays can hold only elements of the same type (homogeneous).	Collections can hold elements of different types (heterogeneous) and the same type.
Have a fixed size and cannot be resized once created.	Have a dynamic size and can grow or shrink as needed.
Not recommended for dynamic memory management due to fixed size.	Recommended for dynamic memory management as they can adjust their size.
Arrays are efficient for direct element access and simple operations.	Collections may have slightly lower performance due to additional abstraction for flexibility.

4. **What is the difference between HashSet and TreeSet?**
   - **HashSet:** Uses a hash table to store elements and does not maintain insertion order.
   - **TreeSet:** Uses a self-balancing binary search tree to store elements and maintains elements in sorted order.
5. **What is the purpose of the Map interface in Java?**

The Map interface in Java is used to store key-value pairs where each key is unique. It allows you to retrieve values based on their associated keys efficiently.

6. **What are the core interfaces of the Java Collection Framework?**

The core interfaces are,

- List
- Set
- Queue

- Map

And their subinterfaces, such as

- SortedSet
- Deque

7. **Explain the difference between ArrayList and LinkedList.**

ArrayList uses an array to store elements and provides fast random access, while LinkedList uses a doubly linked list for storage and offers fast insertion and deletion operations.

8. **Explain the purpose of the Iterator interface.**

Iterator provides a way to traverse through the elements of a collection. It allows sequential access to elements and supports the safe removal of elements during traversal.

9. **How can you sort elements in a Collection in Java?**

You can use the Collections.sort() method to sort elements of a List or utilize SortedSet implementations like TreeSet for automatic sorting based on natural ordering or custom comparators.

10. **How does HashMap work internally in Java?**

HashMap uses an array of linked lists (buckets) to store key-value pairs. The key's hash code determines the index in the array, and collisions are resolved by chaining (using linked lists).

11. **What is the purpose of the Comparable and Comparator interfaces in Java collections?**

Comparable is used for the natural ordering of objects (e.g., sorting Strings alphabetically), while Comparator allows custom ordering of objects based on specific criteria.

12. **Can you explain the differences between HashMap and ConcurrentHashMap?**

HashMap is not thread-safe and not suitable for concurrent access. ConcurrentHashMap, on the other hand, provides thread-safe operations without locking the entire map.

13. **Can you explain the purpose of the Queue interface in Java collections?**

The Queue interface represents a collection designed to hold elements prior to processing. It follows the FIFO (First-In-First-Out) principle.

14. **How would you synchronize access to a non-thread-safe ArrayList in a multithreaded environment?**

You can use Collections.synchronizedList() to wrap an ArrayList and obtain a thread-safe version. Alternatively, use CopyOnWriteArrayList for concurrent read-heavy scenarios.

1. What do you mean by Java Collections?

Ans: A **collection in Java** is a framework that provides an architecture for storing and manipulating a set of objects. JDK 1.2 introduced a new framework called "Collection Framework," which contains all of the collection classes and interfaces.

Collections in Java can perform any data operation, including searching, sorting, inserting, manipulating, and deleting.
A collection in Java refers to a single unit of objects. The Collection interface (java.util.Collection) and the Map interface are the two basic "root" interfaces of Java collection classes (java.util.Map). The Java Collection framework provides numerous

interfaces (Set, Queue, Deque, List) and classes (Vector, ArrayList, LinkedList, PriorityQueue, TreeSet, HashSet, LinkedHashSet).

2. What is the difference between an Array and a Collection in Java?

Ans: Array and Collection are similar in storing object references and manipulating data, but they differ in several ways. The following are the primary differences between an array and a Collection:

Array in Java	Collection in Java
Arrays have a fixed size; we can't adjust them to match our needs once we've built one. A user cannot change the length of the array at runtime or according to their requirements.	Collections are naturally growable and can be modified to match our specific requirements. We can modify the size dynamically to suit our needs.
Arrays can only store elements of the same data type, i.e., homogeneous.	A collection can hold both homogeneous and heterogeneous components.
Arrays outperform Collections when it comes to performance.	When it comes to performance, Collections aren't as good as Arrays.
Arrays can store both objects and primitives.	A collection can only include object types.
There is no ready-made method support for arrays because they have no underlying data structure.	As every collection class is based on a standard data structure, there are ready-made methods for every performance demand. We are not

	responsible for the implementation of these methods, which can be used directly.
Arrays are not preferable to Collections when it comes to memory.	When it comes to memory, Collections outnumber Arrays.

3. What is a collection framework in Java?

Ans: The Collection framework in Java provides a framework for storing and managing a collection of objects. It gives developers access to prepackaged data structures as well as data manipulation algorithms.
The following is part of the collection framework:

- Interfaces
- Classes
- Algorithms

All of these classes and interfaces support various operations such as searching, sorting, inserting, manipulating, and deleting, making data manipulation extremely simple and quick.

4. What are the advantages of the Collection Framework in Java?

Ans: The following are the various advantages of the Java collection framework:

Advantages	Explanation
Performance	Java's collection framework provides highly effective and efficient data structures, which improves a program's speed and accuracy.

Maintainability	Since the collection framework supports data consistency and interoperability within the implementation, the code developed is simple to maintain.
Extensibility	The Java collection framework allows developers to customize the primitive collection types to meet their specific needs.
Reusability	The classes in the collection framework can easily mix with other kinds, increasing code reusability.

5. What are the different interfaces used in the Collection framework?

Ans: The Java Collection Framework implements several interfaces, the most commonly used of which are the Collection and Map interfaces (java.util.Map).

The following is a list of Collection Framework interfaces:
Collection interface: The primary interface is Collection (java.util.Collection), and every collection must implement it.
Syntax: **public interface Collection<A>extends Iterable**
Where <A> denotes that this interface is of type Generic.

List interface: The List interface is an ordered collection of items that extends the Collection interface. It has elements that are duplicated. It also enables element access at random.
Syntax: **public interface List<A> extends Collection<A>**

Map interface: A Map (java.util.Map) is a type of element storage that stores key-value pairs. The Collection interface is not implemented by the Map interface. It can only have one unique key, but duplicate elements are allowed. Map interface and Sorted Map are the two interfaces that implement Map in Java.

Set interface: A collection with the Set (java.util.Set) interface cannot include duplicate elements. It can only include inherited Collection interface methods.
Syntax: **public interface Set<A> extends Collection<A>**

Queue interface: The queue data structure is defined by the queue (java.util.Queue) interface, which holds entries in FIFO order (first-in-first-out).

Syntax: **public interface Queue<A> extends Collection<A>**

De-queue interface: It is a two-ended queue. It enables element insertion and removal from both ends. It embeds both stack and queue attributes, allowing it to perform LIFO (last-in-first-out) stack and FIFO (first-in-first-out) queue operations.
Syntax: **public interface Dequeue<A> extends Queue<A>**

6. What is the difference between an ArrayList and a Vector in Java?

Parameters	ArrayList	Vector
Synchronization	The ArrayList cannot be synchronised.	The vector can be synchronised, i.e., which implies that it can only be accessed by one thread at a time.
Data Growth	If the number of elements in an array reaches its maximum, ArrayList increases the array size by 50%.	When an array's size surpasses its maximum, the vector increases by 100%, doubling the current array size.
Performance	Since ArrayList is not synchronised, it is faster than vector operations.	Since vector operations are synchronised, they are slower (thread-safe). When one thread works on a vector, it gains a lock on it, requiring all other threads working on it to wait until the lock is released before continuing.
Traversal	ArrayList can only use Iterator to traverse its elements.	Vector can use both Enumeration and Iterator to traverse its elements.

7. Distinguish between Collection and Collections.

Ans:
Collection

- Collection in Java(java.util.Collection) is an interface.
- It's used to represent a collection of objects as a single entity.
- It is the Collection framework's root interface.
- It is used to derive the collection framework's data structures.
- It's used to depict a group of distinct objects as a single entity.
- The collection has been an interface with a static method since Java 8. The Interface also has abstract and default methods.

Collections

- Collections in Java(java.util.Collections) is a class.
- It's used to define numerous collection object utility methods.
- The collections aresynchronizedsynchronizedsynchronizedsynchronized a utility class.
- It includes a number of static methods for manipulating data structures.
- It defines a variety of helpful collection-related techniques.
- It only contains static methods.

8. Explain how Java supports linked lists.

Ans: Java supports two forms of linked lists:
A **singly linked list** is a form of data structure. Each node in a singly linked list retains the contents of that node as well as a reference/pointer to the next node in the list. There is no reference or pointer to the previous node stored.
The Doubly Linked List are a form of a linked list that allows traversal in both directions across the data items. This is made possible by each node having two links, one connecting to the next node and the other connecting to the previous node.

9. What are some recommended practices for working with Java Collections?

Ans: Some best practices for using Java Collections are as follows:

Choosing the Right Collection: Before we utilize a collection, we need to make sure it's the right one for the problem we're trying to address. Our software will still operate if we choose the wrong one, but it will be inefficient. If we choose the appropriate one, though, our solution will be considerably easier, and our program will run much faster.

Specifying the Collection's Initial Capacity: Almost all collection classes have an overloaded constructor for determining the collection's initial capacity. For example, if we know how many items will be added to the collection, we may set the initial capacity of a new instance.

Using isEmpty() instead of size() to check if a collection is empty: Instead of finding the collection's size and comparing it to zero, we should use the isEmpty() method. This improves the code's readability.

Using Iterators to Iterate Over Collections: Instead of using a for loop to iterate over the collection elements, we should use iterators. The reason for this is that if another thread tries to alter the collection after the iterator has been built, the iterator may throw ConcurrentModificationException. This protects us from bugs.

Using concurrent collections instead of synchronized wrappers: In multi-threaded applications, we should consider using concurrent collections from the java.util.concurrent package instead of the synchronized wrappers provided by the Collections.synchronizedXXX() methods. Concurrent collections are designed for providing maximum performance in concurrent applications because they use synchronization mechanisms like copy-on-write, compare-and-swap, and particular locks.

Unchecked warnings must be removed: Warnings from the Java compiler that isn't checked should be taken seriously. Any warnings that haven't been checked should be removed as soon as possible.

Generic types should be preferred: We should design new methods with generic parameters in mind, as well as convert existing methods to use type parameters, just as we should with generic types because generic methods are safer and easier to use than non-generic ones. Generic methods are also useful in the development of APIs that are both general and reusable.

10. What are the differences between an ArrayList and a LinkedList?

Ans:
ArrayList

- ArrayList makes use of a dynamic array to store elements internally.
- ArrayList is inefficient for manipulation because it requires too much.
- Element manipulation is slower in ArrayList.
- ArrayList is better for storing and retrieving data.
- It can only function as a List.
- ArrayList allows for random access.
- ArrayList uses less memory because it only stores objects.

LinkedList

- LinkedList makes use of a doubly linked list to store elements internally.
- The manipulation of LinkedList is efficient.
- Elements can be manipulated more quickly on LinkedList.
- To manipulate data, LinkedList is preferable.
- It can function as both a list and a queue.
- The LinkedList does not support random access.
- LinkedList has a higher memory overhead because it stores both the object and its address.

11. What are the differences between an Iterator and a ListIterator?

Ans: Iterator traverses the elements only forward, whereas ListIterator traverses the elements both forward and backward.

Iterator	ListIterator
The Iterator only moves forward through the elements.	ListIterator traverses the elements both backward and forward.
The Iterator can be applied to List, Set, and Queue objects.	ListIterator can only be used in List.
While traversing the collection, the Iterator can only perform ‘remove’ operations.	While traversing the collection, ListIterator can perform ‘add’, ‘remove’, and ‘set’ operations.

12. What are the differences between a HashSet and a TreeSet?

Ans: Both the HashSet and TreeSet implement the Set interface. The distinctions between the two are listed below.

- TreeSet maintains ascending order, whereas HashSet does not.
- TreeSet is implemented by a Tree structure, whereas HashSet is supported by a hash table.
- TreeSet is outperformed by HashSet. In performance, HashSet is better.
- TreeSet is supported by TreeMap, whereas HashSet is supported by HashMap.

13. What are the differences between HashMap and Hashtable?

Hashtable	HashMap

The hashtable is synchronized.	HashMap is not synchronized.
A hashtable cannot have a ‘null’ key or a ‘null’ value.	HashMap can have one or more ‘null’ keys and values.
Hashtable is ‘thread-safe’ and can be shared by multiple threads.	HashMap is not ‘thread-safe’. As a result, it is appropriate for non-threaded applications.
Dictionary Class is inherited by Hashtable.	HashMap derives from AbstractMap.

14. In Java, how are the Collection objects sorted?

Ans: Sorting in Java Collections is accomplished through the use of the Comparable and Comparator interfaces. When the Collections.sort() method is called, the elements are sorted in the natural order specified in the compareTo() method. When the Collections.sort(Comparator) method is used, the objects are sorted based on the Comparator interface's compare() method.

15. What is the distinction between Set and Map?

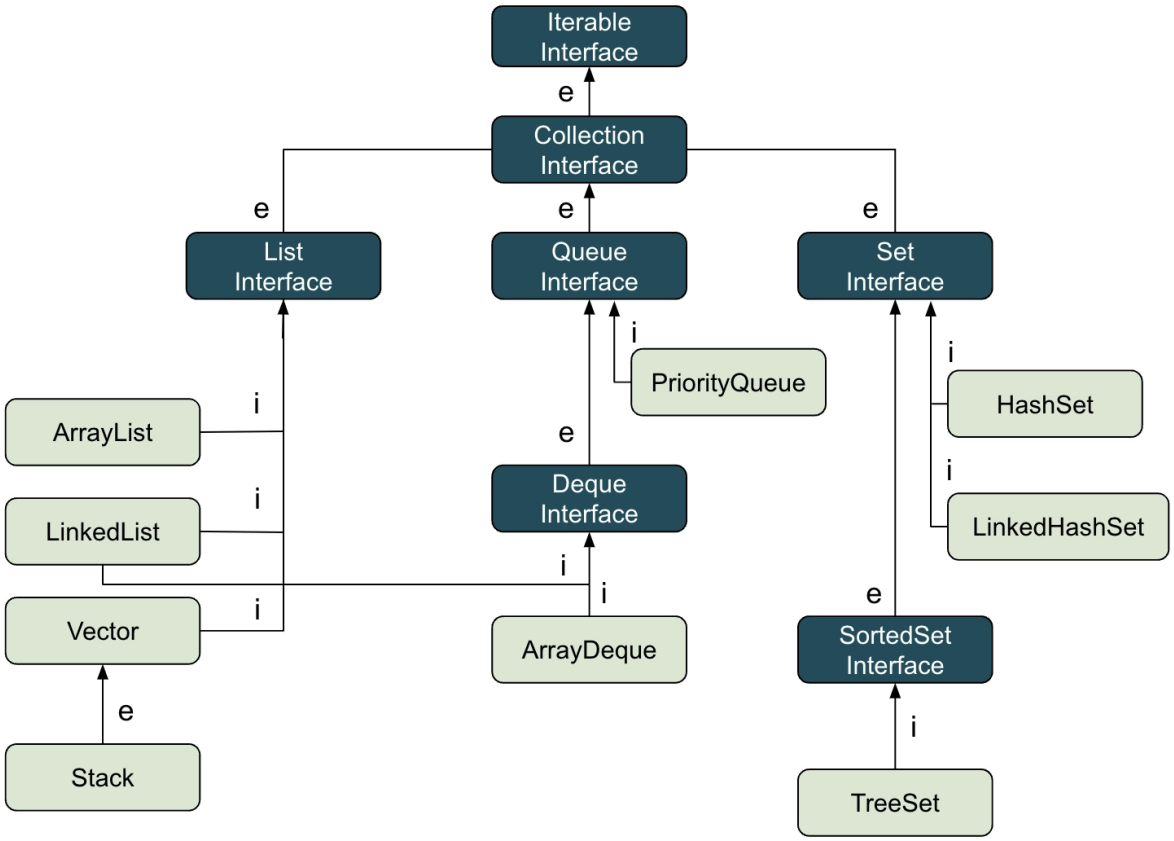
Ans: The distinctions between the Set and the Map are listed below.

- Sets only contain values, whereas Maps contain both keys and values.
- Sets have unique values, whereas Maps can have unique Keys but duplicate values.
- Set has a single number of null values, whereas Map has a single null key with ‘n’ null values.

16. Explain the Collection framework's hierarchical structure in Java.

Ans: The collection framework hierarchy is comprised of four fundamental interfaces: Collection, List, Set, and Map, as well as two sorting-specific interfaces

called SortedSet and SortedMap. The collection framework's interfaces and classes are all contained in java.util package. The Java collection structure is depicted in the diagram below.



Here, ‘e’ stands for extends and ‘i’ stands for implements.
Extends: The extends keyword is used to establish inheritance between two classes and two interfaces.
Implements: The keyword implements is used to define inheritance between classes and interfaces.

17. What is the distinction between a List and a Set?

Ans: The collection interface is extended by the List and Set. However, there are some distinctions between the two, which are detailed below.

- The List can have duplicate elements, whereas the Set has unique items.
- The List is an ordered collection that keeps the insertion order, whereas the Set is an unordered collection that gives up the insertion order.

- The List interface has a single legacy class, i.e., vector, whereas the Set interface does not have any legacy classes.

- The List interface allows 'n' null values, whereas the Set interface only allows one null value.

18. What are the differences between Comparable and Comparator?

Comparable	Comparator
Comparable only provides one type of sequence.	The Comparator provides a variety of sequences.
It has one method called compareTo().	It has one method called compare().
It can be found in the java.lang package.	It is included in the java.util package
When we implement the Comparable interface, the actual class changes.	The actual class remains unchanged.

19. How should one remove duplicates in an ArrayList?

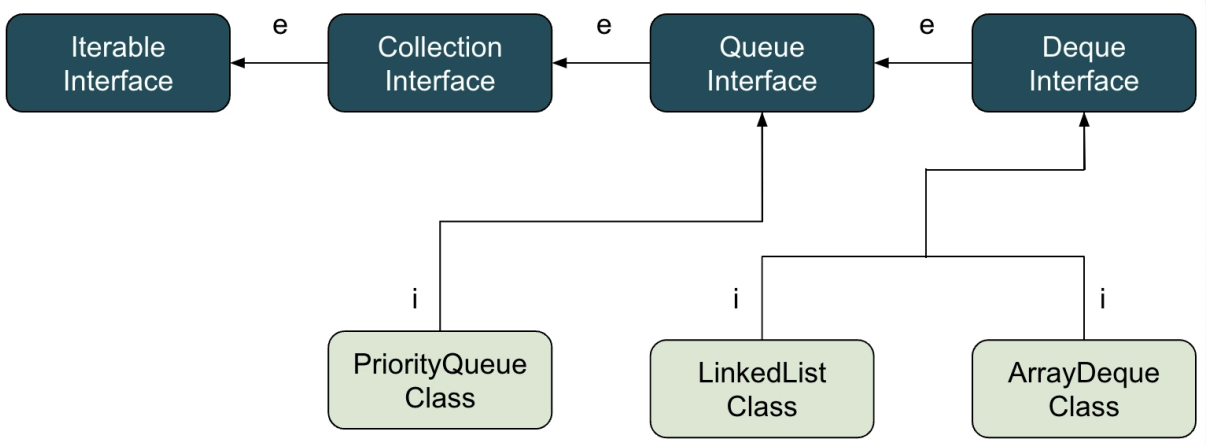
Ans: Duplicates in the ArrayList can be removed in two ways.
Using HashSet: We can remove the duplicate element from the ArrayList using HashSet, but the insertion order will be lost.
Using LinkedHashSet instead of HashSet: We can also keep the insertion order by using LinkedHashSet instead of HashSet.

The LinkedHashSet method is used to remove duplicate elements from an ArrayList. The process of duplicating the elements can be achieved by the following steps:

- ArrayList elements are copied to LinkedHashSet.
- Then clear the ArrayList using the clear() method, which removes all elements from the list.
- Now, copy all of the LinkedHashset elements to ArrayList.

20. What is a priority queue in Java?

Ans: A PriorityQueue is used when the objects are to be processed in order of priority. A Queue is known to use the First-In-First-Out method, but there are times when the queue's components must be handled in order of priority, which is where the PriorityQueue comes in. The PriorityQueue is built on the priority heap. The members of the priority queue are ordered either by natural ordering or by a Comparator provided at the time of queue construction.



Here, ‘e’ stands for extends, and ‘i’ stands for implements.
Extends: The extends keyword is used to establish inheritance between two classes and two interfaces.
Implements: The keyword implements is used to define inheritance between classes and interfaces.
The PriorityQueue class in Java implements the Serializable, Collection<E>, Iterable<E>, and Queue<E> interfaces.

Java Collection Interview Questions For Experienced

21. Distinguish between fail-fast and fail-safe iterators.

Fail-Fast	Fail-Safe
It is not possible to modify the collection while iterating.	It allows collection changes while iterating.
ConcurrentModificationException may be thrown.	It cannot raise any exceptions.
It traverses the elements using the original collection.	It traverses the elements using an original collection copy.
There is no need for additional memory.	There is a requirement for additional memory.

22. Why doesn't the Map interface extend the Collection Interface or vice versa?

Ans: If Map extends the Collection Interface, "**Key-value pairs**" may be the only element type for this type of Collection, though this provides a very limited (and ineffective) Map abstraction. You can't ask what value a particular key corresponds to, and you can't delete an entry unless you know what value it corresponds to. Maps' three "Collection view procedures" depict the fact that Maps can be viewed as the Collections (of keys, values, or pairs) (**keySet, entrySet, and valueSet**). While it is theoretically possible to think of a List as a Map mapping indices to items, doing so has the unintended consequence of changing the Key associated with every element in the List prior to the deleted member. This is also why the Collection cannot be used to extend the Map Interface.

23. Why do we override the equals() method?

Ans: The equals method is used to determine whether two objects are identical. If we want to check the objects based on the property, we must override it.
For example, Employee is a class that has 3 data members: id, name, and salary. However, we want to test the equality of employee objects based on their salaries. The equals() method must then be overridden.

24. List the differences between an Array and an ArrayList in Java.

Ans: The differences between an Array and an ArrayList are:

- Arrays are a fundamental feature of Java. ArrayList is a collection system component in Java. As a result, It is used to access array members, whereas ArrayList provides a set of methods for modifying and accessing components.

- Although ArrayList is not a fixed-size data structure, Array is. There is no need to specify the size of an ArrayList object when creating it. Even if we set a maximum capacity, we can later add more parts.

- Arrays can contain both primitive data types and class objects, depending on how they are defined. In contrast, ArrayList only accepts object entries and not primitive data types. The primitive int data type is converted to an Integer object when we use arraylist.add(1).

- Since ArrayList cannot be constructed for primitive data types, its members are always referencing objects at different memory locations. As a result, the actual objects in an ArrayList are never stored in the same location. The references to actual items are kept close together. The type of array determines whether it is a primitive or an object. Actual values for primitive types are continuous regions, while object allocation is equivalent to ArrayList.

- Java ArrayList supports a wide range of other operations, including indexOf() and delete(). These functions are not supported by arrays.

25. What do you mean by hash-collision in Hashtable, and how does it work in Java?

Ans: Hash-collision occurs when two keys with the same hash value collide. Two separate entries will be kept in a single hash bucket to avoid collision.
There are two methods for avoiding hash collisions.

- Open Addressing
- Separate Chaining

26. Can we modify the overridden method's scope in the subclass?

Yes, the scope of the overridden method in the subclass can be changed. However, we must keep in mind that we cannot reduce the method's accessibility. The following consideration must be made while modifying the method's accessibility:

- Change the private access specifier to protected, public, or default.
- The protected access specifier can be set to public or default.
- The default setting can be changed to public.
- The public will always remain public.

27. What is method overriding in Java?

Method overriding occurs when a subclass provides a specific implementation of a method that is already offered by its parent class. It is used to implement interface methods and for runtime polymorphism. There are some rules which are necessary for method overriding Java:

- The method's name must be the same as in the parent class.
- The method's signature must be the same as in the parent class.
- An IS-A relationship must exist between two classes.

28. What is composition?

Composition is the holding of a class's reference within another class. When one item includes another, and the contained object cannot exist without the existence of the container object, this is referred to as composition. In other words, composition is a subset of aggregation that represents a stronger relationship between two items. Students, for example, make up a class. A student cannot exist in the absence of a class. There is a composition between the class and the pupils.

29. What are the advantages of Inheritance in Java?

The following are some of the benefits of using inheritance in Java:

- Inheritance allows for code reuse. The derived class is not required to redefine the base class's method unless it needs to provide a specific implementation of the method.
- Inheritance is required to create runtime polymorphism.
- Data hiding is provided by inheritance. By making some data private, the base class can hide it from the descendant class.
- Without inheritance, method overriding is impossible. We can provide a specific implementation of a basic method contained by the base class through method overriding.

30. What is the static variable?

The static variable is used to refer to a property that is shared by all objects but is not unique to each one, such as the firm name of employees, the college name of students, and so on. Static variables are allocated memory in the class region just once during class loading. Using a static variable improves the memory efficiency of your program (it saves memory). The static variable belongs to the class, not the object.

31. How does ConcurrentHashMap work in Java?

ConcurrentHashMap is designed for high concurrency by partitioning the map into segments. Each segment is a separate lockable entity, allowing multiple threads to access different segments simultaneously without interference. This design avoids locking the entire map, as seen in Hashtable, thereby improving performance in multithreaded environments. The map uses fine-grained locking, which reduces contention and enhances throughput by allowing concurrent reads and writes.

32. What is the role of NavigableMap in Java Collections?

NavigableMap extends SortedMap and adds methods for efficient navigation within the map. It allows traversal in both ascending and descending order through methods such as lowerKey(), floorKey(), ceilingKey(), and higherKey(). These methods help locate specific entries relative to a given key, facilitating range queries and other operations that require sorted data.

33. How do WeakHashMap and HashMap differ?

WeakHashMap and HashMap differ in how they manage key references, impacting their memory handling and use cases. WeakHashMap uses weak references for its keys, which means that if a key in the map is no longer reachable through strong references, the garbage collector can reclaim the memory associated with that key. This behavior makes WeakHashMap ideal for memory-sensitive caching scenarios where you want unused entries to be automatically removed to free up memory. In contrast, HashMap uses strong references for its keys, meaning that as long as the key is present in the map, it will not be garbage collected, even if no other references to the key exist. This can lead to memory leaks if keys are not explicitly removed from the map. Thus, while WeakHashMap is beneficial for implementing caches where keys should be cleaned up when not in use, HashMap is more suited for scenarios where you need to maintain the keys in memory until you manually manage their removal.

34. What is the purpose of the BlockingQueue interface?

he BlockingQueue interface is used for thread-safe queue operations where threads may need to wait for certain conditions. It provides methods that support waiting for the queue to become non-empty when retrieving an element and for the queue to have available space when adding an element. This is particularly useful in scenarios like the producer-consumer problem, where you need to manage threads that produce and consume items asynchronously.

35. How does TreeMap maintain order?

TreeMap maintains order through the use of a Red-Black tree, which is a self-balancing binary search tree. This tree structure keeps the elements of the TreeMap sorted according to their natural ordering, such as numerical or alphabetical order, or according to a comparator provided when the TreeMap is created. The Red-Black tree ensures that the map remains balanced by performing rotations and color changes during insertions and deletions. This balanced structure allows TreeMap to provide efficient performance for common operations. For example, searching, inserting, and deleting elements all have a time complexity of O(log n), ensuring that the elements are kept in a sorted order while maintaining quick access times. By balancing the tree, TreeMap can offer consistent and predictable performance while managing a sorted collection of key-value pairs.

36. Explain the concept of CopyOnWriteArrayList.

CopyOnWriteArrayList is a thread-safe variant of ArrayList designed for situations where read operations are more frequent than write operations. It works by creating a new copy of the underlying array for every write operation, such as adding or

removing elements. This strategy ensures that readers access a stable snapshot of the list while writes are applied to a new copy, thus avoiding synchronization issues and making it suitable for concurrent environments.

37. What is the difference between Synchronized Collection and Concurrent Collection?

Synchronized Collections are thread-safe wrappers around standard collections like ArrayList or HashMap, achieved by synchronizing each method call, which can introduce performance overhead due to the locking mechanism. Concurrent Collections, such as ConcurrentHashMap and CopyOnWriteArrayList, are designed with concurrent access in mind, offering better performance and scalability by using advanced concurrency techniques like fine-grained locking or lock-free algorithms, reducing contention among threads.

38. How does the PriorityBlockingQueue work in Java?

The PriorityBlockingQueue in Java is a thread-safe, unbounded queue that sorts its elements based on their natural ordering or a comparator specified during its creation. It uses a priority heap to manage the order of elements, ensuring that the highest-priority element is always at the front of the queue. Unlike some other queues, PriorityBlockingQueue does not block when adding elements, meaning it can accept new entries without waiting.

However, it does block on retrieval operations if the queue is empty, making it ideal for scenarios where threads need to wait for tasks to become available. This combination of features makes PriorityBlockingQueue suitable for managing tasks in concurrent environments where the priority of tasks needs to be respected while also handling situations where task availability can be intermittent.

39. What are Phantom References in Java, and how do they relate to collections?

Phantom References in Java are a type of reference that allows you to track the lifecycle of objects in memory without preventing their garbage collection. Unlike soft or weak references, phantom references do not give you access to the referenced object. Instead, they are used in conjunction with a ReferenceQueue to receive notifications when an object becomes unreachable. This is particularly useful for implementing memory-sensitive caching mechanisms and performing cleanup operations before an object is actually removed from memory.

40. What is IdentityHashMap in Java?

IdentityHashMap is a unique implementation of the Map interface in Java that uses reference equality (==) instead of the standard object equality (equals) for comparing keys. This means that two keys are considered equal only if they are the exact same object in memory, not if they are equal according to their equals method.
This behavior is particularly useful in situations where you need to ensure that keys are unique based on their memory address, not their content. For example, if you have a situation where the exact identity of objects is important, such as managing a set of unique objects where their identity must be tracked directly, IdentityHashMap is the appropriate choice. It helps when reference equality is crucial, and you want to avoid potential issues that can arise from object equality checks.

1. What are the advantages of the Collection Framework in Java?

Below table contains the major advantages of the Java Collection

Framework:

Feature	Description
Performance	The collection framework provides highly effective and efficient data structures that result in enhancing the speed and accuracy of a program.
Maintainability	The code developed with the collection framework is easy to maintain as it supports data consistency and interoperability within the implementation.
Reusability	The classes in Collection Framework can effortlessly mix with other types which results in increasing the code reusability.
Extensibility	The Collection Framework in Java allows the developers to customize the primitive collection types as per their requirements.

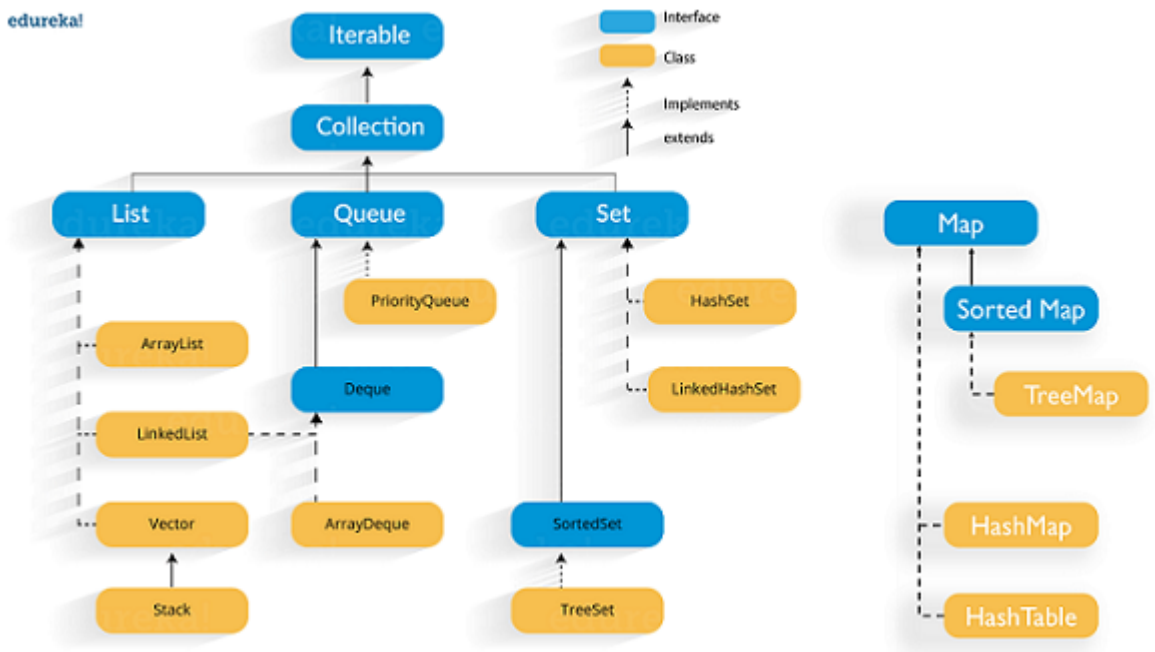
2. What do you understand by Collection Framework in Java?

The Java Collection framework provides an architecture to store and manage a group of objects. It permits the developers to access prepackaged data structures as well as algorithms to manipulate data. The collection framework includes the following:

- Interfaces
- Classes
- Algorithm

All these classes and interfaces support various operations such as Searching, Sorting, Insertion, Manipulation, and Deletion which makes the data manipulation really easy and quick.

3. Describe the Collection hierarchy in Java.



4. List down the primary interfaces provided by Java Collections Framework?

Below are the major interfaces provided by the Collection Framework:

- ***Collection Interface*:** java.util.The collection is the root of the Java Collection framework and most of the collections in Java are inherited from this interface.

```
public interface Collection<E>extends Iterable
```

- ***List Interface*:** java.util.List is an extended form of an array that contains ordered elements and may include duplicates. It supports the index-based search, but elements can be easily inserted irrespective of the position. The List interface is implemented by various classes such as ArrayList, LinkedList, Vector, etc.

```
public interface List<E> extends Collection<E>
```

- ***Set Interface*:** java.util.Set refers to a collection class that cannot contain duplicate elements. Since it doesn't define an order for the elements, the index-based search is not supported. It is majorly used as a mathematical set abstraction model. The Set interface is implemented by

various classes such as HashSet, TreeSetand LinkedHashSet.

```
public interface Set<E> extends Collection<E>
```

- ***Queue Interface*:** java.util.Queue in Java follows a FIFO approach i.e. it orders the elements in First In First Out manner. Elements in Queue will be
- ***Map Interface*:** java.util.Map is a two-dimensional data structure in Java that is used to store the data in the form of a Key-Value pair. The key here is the unique hashcode and value represent the element. Map in Java is another form of the Java Set but can't contain duplicate elements.

5. Why Collection doesn't extend the Cloneable and Serializable interfaces?

The Collection interface in Java specifies a group of objects called elements. The maintainability and ordering of elements are completely dependent on the concrete implementations provided by

each of the Collection. Thus, there is no use of extending the Cloneable and Serializable interfaces.

6. List down the major advantages of the Generic Collection.

Below are the main advantages of using the generic collection in Java:

- Provides stronger type checks at the time of compilation
- Eliminates the need for typecasting
- Enables the implementation of generic algorithms which makes the code customizable, type-safe and easier to read

7. What is the main benefit of using the Properties file?

The main advantage of using the properties file in Java is that in case the values in the properties file are changed it will be automatically reflected without having to recompile the java class. Thus it is mainly used to store information that is liable to change

such as username and passwords. This makes the management of the application easy and efficient. Below is an example of the same:

```
import java.util.*;

import java.io.*;

public class PropertiesDemo{

public static void main(String[] args)throws Exception{

FileReader fr=new FileReader("db.properties");

Properties pr=new Properties();

pr.load(fr);

System.out.println(pr.getProperty("user"));

System.out.println(pr.getProperty("password"));

}

}
```

8. What do you understand by the Iterator in the Java Collection Framework?

Iterator in Java is an interface of the Collection framework present in java.util package. It is a Cursor in Java which is used to iterate a collection of objects. Below are a few other major functionalities provided by the Iterator interface:

- Traverse a collection object elements one by one
- Known as Universal Java Cursor as it is applicable for all the classes of the Collection framework
- Supports READ and REMOVE Operations.
- Iterator method names are easy to implement

9. What is the need for overriding equals() method in Java?

The initial implementation of the equals method helps in checking whether two objects are the same or not. But in case you want to compare the objects based on the property you will have to override this method.

10. How the Collection objects are sorted in Java?

Sorting in Java Collections is implemented via Comparable and Comparator interfaces. When Collections.sort() method is used the elements get sorted based on the natural order that is specified in the compareTo() method. On the other hand when Collections.sort(Comparator) method is used it sorts the objects based on compare() method of the Comparator interface.

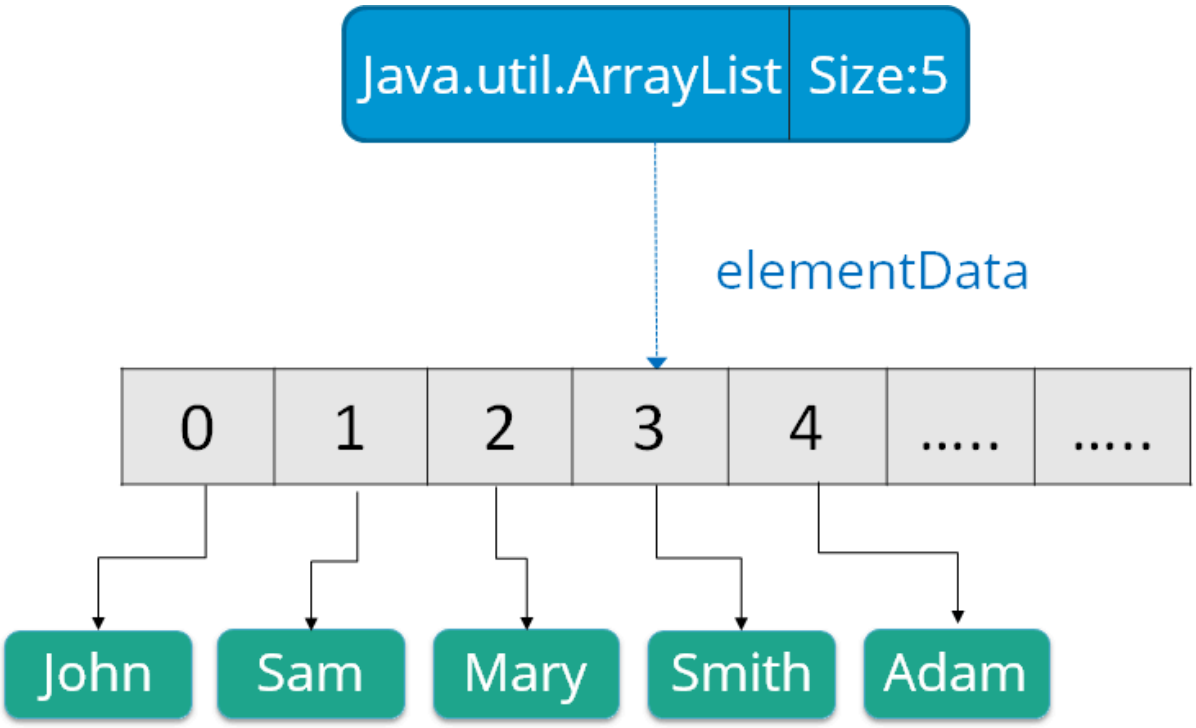
List — Java Collections Interview Questions

11. What is the use of the List interface?

The List interface in Java is an **ordered collection** of elements. It maintains the insertion order and allows duplicate values to be stored within. This interface contains various methods that enable smooth manipulation of elements based on the element index. The main classes implementing the List interface of the Collection framework are **ArrayList**, **LinkedList**, **Stack, and Vector**.

12. What is ArrayList in Java?

ArrayList is the implementation of List Interface where the elements can be dynamically added or removed from the list. ArrayList in the Collection framework provides positional access and insertion of elements. It is an ordered collection that permits duplicate values. The size of an ArrayList can be increased dynamically if the number of elements is more than the initial size.



Syntax:

```
ArrayList object = new ArrayList ();
```

13. How would you convert an ArrayList to Array and an Array to ArrayList?

An Array can be converted into an ArrayList by making use of the asList() method provided by the Array class. It is a static method that accepts List objects as a parameter.

Syntax:

```
Arrays.asList(item)
```

Whereas an ArrayList can be converted into an array using the toArray() method of the ArrayList class.

Syntax:

```
List_object.toArray(new String[List_object.size()])
```

14. How will you reverse a List?

ArrayList can be reversed using the reverse() method of the Collections class.

Syntax:

```
public static void reverse(Collection c)
```

For Example:

```
public class ReversingArrayList {

public static void main(String[] args) {

List<String> myList = new ArrayList<String>();

myList.add("AWS");

myList.add("Java");

myList.add("Python");

myList .add("Blockchain");

System.out.println("Before Reversing");

System.out.println(myList.toString());
```

```
Collections.reverse(myList);

System.out.println("After Reversing");

System.out.println(myList);

}

}
```

15. What do you understand by LinkedList in Java? How many types of LinkedList does Java support?

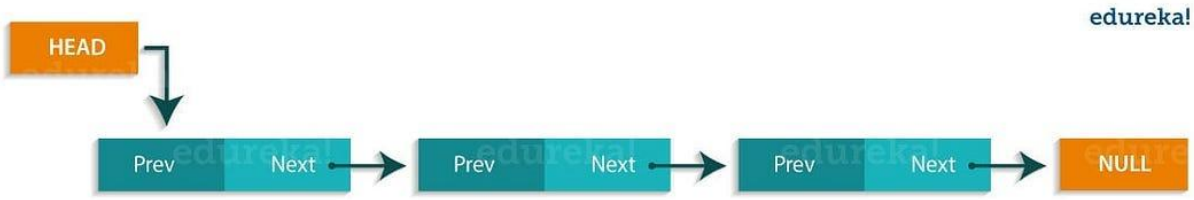
LinkedList in Java is a data structure that contains a sequence of links. Here each link contains a connection to the next link.

Syntax:

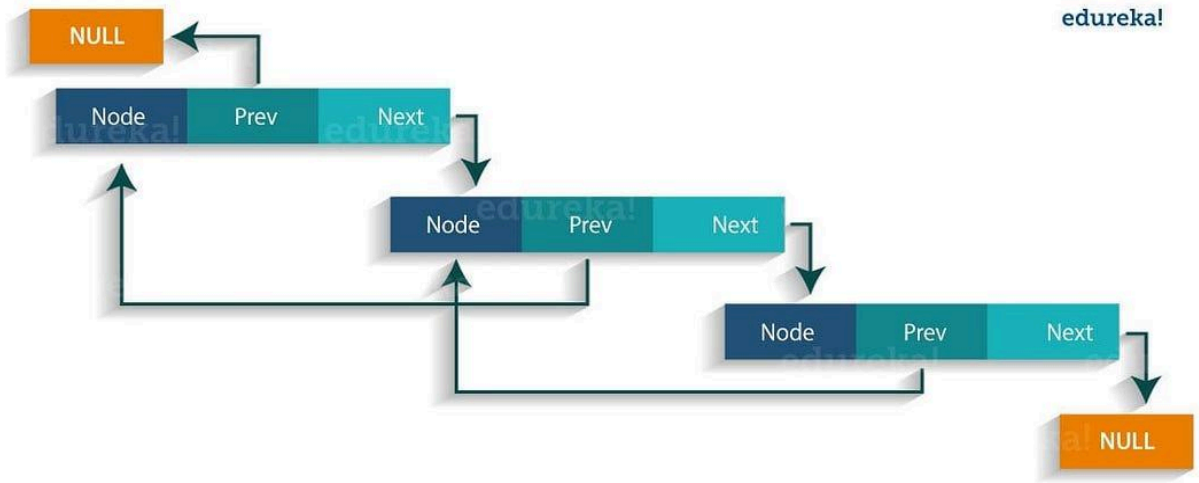
```
Linkedlist object = new Linkedlist();
```

Java LinkedList class uses two types of LinkedList to store the elements:

- ***Singly Linked List:*** In a singly LinkedList, each node in this list stores the data of the node and a pointer or reference to the next node in the list.



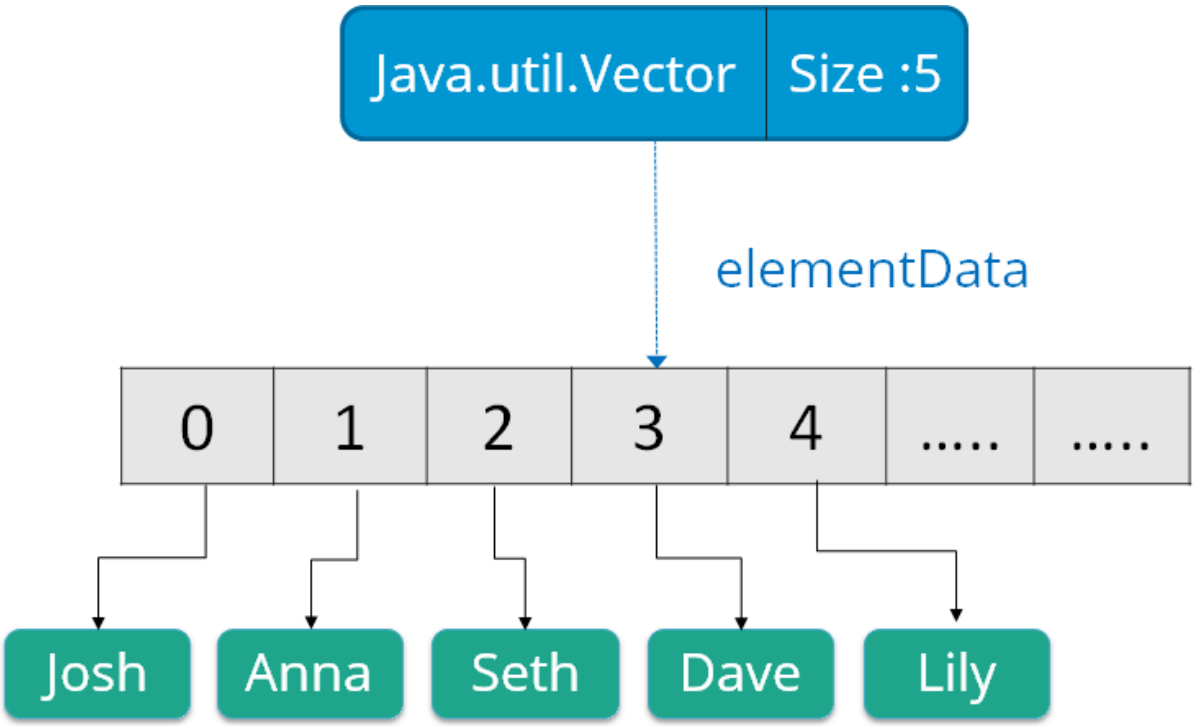
- ***Doubly Linked List:*** In a doubly LinkedList, it has two references, one to the next node and another to the previous node.



16. What is a Vector in Java?

Vectors are similar to arrays, where the elements of the vector object can be accessed via an index into the vector. Vector implements a dynamic array. Also, the vector is not limited to a specific size, it can shrink or grow automatically whenever required. It is similar to ArrayList, but with two differences :

- Vector is synchronized.
- The vector contains many legacy methods that are not part of the collections framework.



```
Vector object = new Vector(size,increment);
```

Queue — Java Collections Interview Questions

17. What are the various methods provided by the Queue interface?

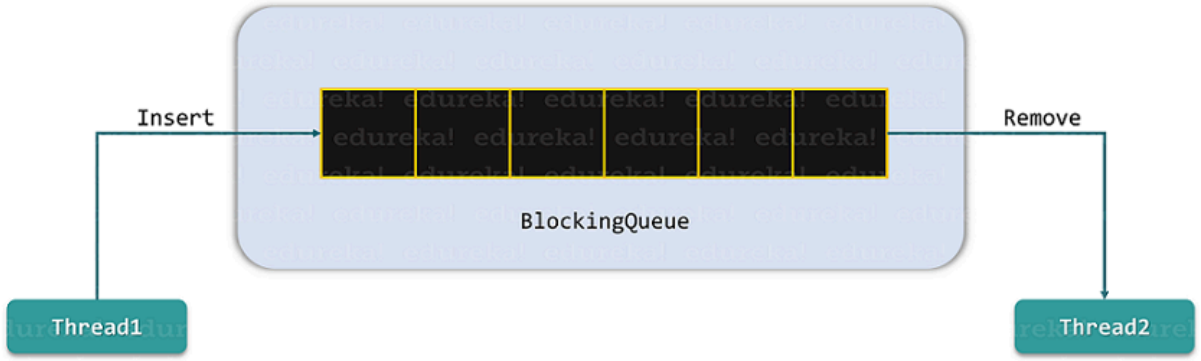
Below are some of the methods of Java Queue interface:

Method	Description
boolean add(object)	Inserts the specified element into the queue and returns true if it is a success.
boolean offer(object)	Inserts the specified element into this queue.
Object remove()	Retrieves and removes the head of the queue.
Object poll()	Retrieves and removes the head of the queue, or returns null if the queue is empty.
Object element()	Retrieves, but does not remove the head of the queue.
Object peek()	Retrieves, but does not remove the head of this queue, or returns null if the queue is empty.

18. What do you understand by BlockingQueue?

BlockingQueue interface belongs to the **java.util.concurrent** package. This interface enhances flow control by activating

blocking, in case a thread is trying to dequeue an empty queue or enqueue an already full queue. While working with the BlockingQueue interface in Java, you must remember that it does not accept a null value. In case you try to do that it will instantly throw a NullPointerException. The below figure represents the working of the BlockingQueue interface in Java.



19. What is a priority queue in Java?

A priority queue in Java is an abstract data type similar to a regular queue or stack data structure but has a special feature called priority associated with each element. In this queue, a high priority element is served before a low priority element irrespective of their insertion order. The PriorityQueue is based on the priority heap.

The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.

20. What is the Stack class in Java and what are the various methods provided by it?

Java Stack class is an important part of the Java Collection framework and is based on the basic principle of last-in-first-out. In other words, the elements are added as well as removed from the rear end. The action of adding an element to a stack is called push while removing an element is referred to as pop. Below are the various methods provided by this class:

Methods	Description
empty()	Checks if the stack is empty
push()	Push an item to the top of the stack
pop()	Remove the object from the stack
peek()	Looks at the object of a stack without removing it
search()	Searches item in the stack to get its index

Set — Java Collections Interview Questions

21. What is Set in Java Collections framework and list down its various implementations?

A Set refers to a collection that cannot contain duplicate elements. It is mainly used to model the mathematical set abstraction. The Java platform provides three general-purpose Set implementations which are:

1. HashSet
2. TreeSet
3. LinkedHashSet

22. What is the HashSet class in Java and how does it store elements?

java.util.HashSet class is a member of the Java collections framework which inherits the AbstractSet class and implements the Set interface. It implicitly implements a hashtable for creating and storing a collection of unique elements. Hashtable is an instance of the HashMap class that uses a hashing mechanism for storing the

information within a HashSet. Hashing is the process of converting the informational content into a unique value that is more popularly known as hash code. This hashcode is then used for indexing the data associated with the key. The entire process of transforming the informational key into the hashcode is performed internally.

23. Can you add a null element into a TreeSet or HashSet?

In HashSet, only one null element can be added but in TreeSet it can’t be added as it makes use of NavigableMap for storing the elements. This is because the NavigableMap is a subtype of SortedMap that doesn’t allow null keys. So, in case you try to add null elements to a TreeSet, it will throw a NullPointerException.

24. Explain the emptySet() method in the Collections framework?

The Collections.emptySet() is used to return the empty immutable Set while removing the null elements. The set returned by this

method is serializable. Below is the method declaration of emptySet().

Syntax:

```
public static final <T> Set<T> emptySet()
```

25. What is LinkedHashSet in Java Collections Framework?

A java.util.LinkedHashSet is a subclass of the HashSet class and implements the Set interface. Itis an ordered version of HashSet which maintains a doubly-linked List across all elements contained within. It preserves the insertion order and contains only unique elements like its parent class.

Syntax:

```
LinkedHashSet<String> hs = new LinkedHashSet<String>();
```

Map — Java Collections Interview Questions

26. What is Map interface in Java?

The java.util.Map interface in Java stores the elements in the form of keys-values pairs which is designed for faster lookups. Here every key is unique and maps to a single value. These key-value pairs are known as the map entries. This interface includes method signatures for insertion, removal, and retrieval of elements based on a key. With such methods, it's a perfect tool to use for key-value association mapping such as dictionaries.

27. Why Map doesn't extend the Collection Interface?

The Map interface in Java follows a key/value pair structure whereas the Collection interface is a collection of objects which are stored in a structured manner with a specified access mechanism. The main reason Map doesn't extend the Collection interface is that the add(E e) method of the Collection interface doesn't support the key-value pair like Map interface's put(K, V) method. It might not extend the Collection interface but still is an integral part of the Java Collections framework.

28. List down the different Collection views provided by the Map interface in the Java Collection framework?

The Map interface provides 3 views of key-value pairs which are:

- key set view
- value set view
- entry set view

All these views can be easily navigated through using the iterators.

29. What is the ConcurrentHashMap in Java and do you implement it?

ConcurrentHashMap is a Java class that implements ConcurrentMap as well as to Serializable interfaces. This class is the enhanced version of HashMap as it doesn't perform well in the multithreaded environment. It has a higher performance rate compared to the HashMap.

Below is a small example demonstrating the implementation of ConcurrentHashMap:

```
package edureka;

import java.util.concurrent.*;



public class ConcurrentHashMapDemo {

   public static void main(String[] args)

   {

        ConcurrentHashMap m = new ConcurrentHashMap();

        m.put(1, "Welcome");

        m.put(2, "to");

        m.put(3, "Edureka's");

        m.put(4, "Demo");
```

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




        // Here we cant add Hello because 101 key

        // is already present in ConcurrentHashMap object

        m.putIfAbsent(3, "Online");

        System.out.println("Checking if key 3 is already present in the
ConcurrentHashMap object: "+ m);




        // We can remove entry because 101 key

        // is associated with For value

        m.remove(1, "Welcome");

        System.out.println("Removing the value of key 1: "+m);
```

```
        // Now we can add Hello

        m.putIfAbsent(1, "Hello");

        System.out.println("Adding new value to the key 1: "+m);



        // We cant replace Hello with For

        m.replace(1, "Hello", "Welcome");

        System.out.println("Replacing value of key 1 with Welcome: "+
m);

    }

}
```

30. Can you use any class as a Map key?

Yes, any class can be used as Map Key as long as the following points are considered:

- The class overriding the equals() method must also override the hashCode() method
- The class should adhere to the rules associated with equals() and hashCode() for all instances
- The class field which is not used in the equals() method should not be used in hashCode() method as well
- The best way to use a user-defined key class is by making it immutable. It helps in caching the hashCode() value for better performance. Also if the class is made immutable it will ensure that the hashCode() and equals() are not changing in the future.

Differences — Java Collections Interview Questions

31. Differentiate between Collection and Collections.

Collection	Collections
java.util.Collection is an interface	java.util.Collections is a class
Is used to represent a group of objects as a single entity	It is used to define various utility method for collection objects
It is the root interface of the Collection framework	It is a utility class
It is used to derive the data structures of the Collection framework	It contains various static methods which help in data structure manipulation

32. Differentiate between an Array and an ArrayList.

Array	ArrayList
java.util.Array is a class	java.util.ArrayList is a class
It is strongly typed	It is loosely types
Cannot be dynamically resized	Can be dynamically resized
No need to box and unbox the elements	Needs to box and unbox the elements

33. Differentiate between Iterable and Iterator.

Iterable	Iterator
Iterable is an interface	Iterator is an interface
Belongs to java.lang package	Belongs to java.util package
Provides one single abstract method called iterator()	Provides two abstract methods called hasNext() and next()
It is a representation of a series of elements that can be traversed	It represents the object with iteration state

34. Differentiate between ArrayList and LinkedList.

ArrayList	LinkedList
Implements dynamic array internally to store elements	Implements doubly linked list internally to store elements
Manipulation of elements is slower	Manipulation of elements is faster
Can act only as a List	Can act as a List and a Queue
Effective for data storage and access	Effective for data manipulation

35. Differentiate between Comparable and Comparator.

Comparable	Comparator
Present in java.lang package	Present in java.util package
Elements are sorted based on natural ordering	Elements are sorted based on user-customized ordering
Provides a single method called compareTo()	Provides to methods equals() and compare()
Modifies the actual class	Doesn't modifies the actual class

36. Differentiate between List and Set.

List	Set
An ordered collection of elements	An unordered collection of elements
Preserves the insertion order	Doesn't preserves the insertion order
Duplicate values are allowed	Duplicate values are not allowed
Any number of null values can be stored	Only one null values can be stored

37. Differentiate between Set and Map.

Set	Map
Belongs to java.util package	Belongs to java.util package
Extends the Collection interface	Doesn't extend the Collection interface
Duplicate values are not allowed	Duplicate keys are not allowed but duplicate values are
Only one null values can be stored	Only one null key can be stored but multiple null values are allowed

38. Differentiate between List and Map.

List	Map
Belongs to java.util package	Belongs to java.util package
Extends the Collection interface	Doesn't extend the Collection interface
Duplicate elements are allowed	Duplicate keys are not allowed but duplicate values are
Multiple null values can be stored	Only one null key can be stored but multiple null values are allowed

39. Differentiate between Queue and Stack.

Queue	Stack
Based on FIFO (First-In-First-Out) principle	Based on LIFO (Last-In-First-Out) principle
Insertion and deletion takes place from two opposite ends	Insertion and deletion takes place the same end
Element insertion is called enqueue	Element insertion is called push
Element deletion is called dequeue	Element deletion is called pop

40. Differentiate between PriorityQueue and TreeSet.

Priority Queue	Tree Set
It is a type of Queue	It is based on a Set data structure
Allows duplicate elements	Doesn't allows duplicate elements
Stores the elements based on an additional factor called priority	Stores the elements in a sorted order

41. Differentiate between the Singly Linked List and Doubly Linked List.

Singly Linked List(SLL)	Doubly Linked List(DLL)
Contains nodes with a data field and a next node-link field	Contains nodes with a data field, a previous link field, and a next link field
Can be traversed using the next node-link field only	Can be traversed using the previous node-link or the next node-link
Occupies less memory space	Occupies more memory space
Less efficient in providing access to the elements	More efficient in providing access to the elements

42. Differentiate between Iterator and Enumeration.

Iterator	Enumeration
Collection element can be removed while traversing it	Can only traverse through the Collection
Used to traverse most of the classes of the Java Collection framework	Used to traverse the legacy classes such as Vector, HashTable, etc
Is fail-fast in nature	Is fail-safe in nature
Is safe and secure	Is not safe and secure

43. Differentiate between HashMap and HashTable.

HashMap	HashTable
It is non-synchronized in nature	It is synchronized in nature
Allows only one null key but multiple null values	Doesn't allow any null key or value
Has faster processing	has slower processing
Can be traversed by Iterator	Can be traversed by Iterator and Enumeration

44. Differentiate between HashSet and HashMap.

HashSet	HashMap
Based on Set implementation	Based on Map implementation
Doesn't allow any duplicate elements	Doesn't allow any duplicate keys but duplicate values are allowed
Allows only a single null value	Allows only one null key but any number of null values
Has slower processing time	Has faster processing time

45. Differentiate between Iterator and ListIterator.

Iterator	ListIterator
Can only perform remove operations on the Collection elements	Can perform add, remove and replace operations the Collection elements
Can traverse List, Sets and maps	Can traverse only Lists
Can traverse the Collection in forward direction	Can traverse the collection in any direction
Provides no method to retrieve the index of the element	Provides methods to retrieve the index of the elements

46. Differentiate between HashSet and TreeSet.

HashSet	TreeSet
Uses HasMap to store elements	Uses Treemap to store elements
It is unordered in nature	By default, it stores elements in their natural ordering
Has faster processing time	Has slower processing time
Uses hasCode() and equals() for comparing	Uses compare() and compareTo() for comparing

47. Differentiate between Queue and Deque.

Queue	Deque
Refers to single-ended queue	Refers to double-ended queue
Elements can be added or removed from only one end	Elements can be added and removed from either end
Less versatile	More versatile

48. Differentiate between HashMap and TreeMap.

HashMap	TreeMap
Doesn't preserves any ordering	Preserves the natural ordering
Implicitly implements the hashing principle	Implicitly implements the Red-Black Tree Implementation
Can store only one null key	Cannot store any null key

49. Differentiate between ArrayList and Vector.

ArrayList	Vector
Non-synchronized in nature	Synchronized in nature
It is not a legacy class	Is a legacy class
Increases size by 1/2 of the ArrayList	Increases size by double of the ArrayList
It is not thread-safe	It is thread-safe

50. Differentiate between failfast and failsafe.

failfast	failsafe
Doesn’t allow modifications of a collection while iterating	Allows modifications of a collection while iterating
Throws ConcurrentModificationException	Don’t throw any exceptions
Uses the original collection to traverse over the elements	Uses a copy of the original collection to traverse over the elements
Don’t require extra memory	Require extra memory