## Why Collection in Java so important?

**1) Data Organization**: Collections allow us to organize and store data effectively. They provide data structures for managing groups of objects, such as lists, sets, and maps, which are fundamental in many programming scenarios.

**2) Code Reusability**: Collections provide a set of well-implemented data structures. We can use these data structures to manage our data successfully, leading to more reusable and cleaner code.

**3) Data Retrieval and Manipulation**: Collections offer various methods for retrieving and manipulating data. We can use this methods such as add, remove, iterate, and search for elements within collections effortlessly, which simplifies data processing tasks.

**4) Performance**: Proper use of collections can lead to better performance. The choice of the right collection for the task can significantly impact the efficiency of our code. For example, lists are suitable for ordered data, while sets are useful for maintaining unique values.

**5) API Familiarity**: Learning collections is worthful because they are part of the Java API, and many Java libraries and frameworks use them extensively. Being proficient in collections makes it easier to work with Java libraries and understand third-party code.

**6)** **Concurrency and Thread Safety**: Understanding concurrent collections is essential for writing multithreaded applications. Java provides concurrent collections designed for safe access from multiple threads. It helps us build robust concurrent programs.

## Top 20 Collection Framework In Java Interview Questions & Answers

1. **What is the difference between a stack and a queue in Java Collections?**

A stack is a collection in Java that stores elements in a last-in-first-out (LIFO) order, while a queue stores elements in a first-in-first-out (FIFO) order. In a stack, LIFO means elements are pushed onto the top and popped from the top. In a queue, FIFO means elements are added to the rear and removed from the front.

1. **When should we use CopyOnWriteArrayList Collection?**

When the number of read operations is considerably more than the number of write operations, the CopyOnWriteArrayList may be the best choice to use. Since it offers thread-safety for concurrent reads, therefore it can be ideal for scenarios where data consistency is conclusive. However, the [CopyOnWriteArrayList](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) is not suitable for scenarios where frequent modifications or updates are required, as each modification involves creating a new copy of the entire list. Hence, it can be memory-intensive and impact performance.

1. **What is the difference between a Vector and an ArrayList?**

A Vector is a thread-safe implementation of an ArrayList in Java Collections. It allows effective concurrent access and modification by multiple threads. An ArrayList is not thread-safe, and requires explicit synchronization to enable thread-safety. However, the use of Vectors is not recommended in modern Java programming due to their inefficiency compared to other thread-safe collections.

1. **What is the difference between a IdentityHashMap and a HashMap in Java Collections?**

An IdentityHashMap is a collection in Java that uses reference equality (== operator) rather than object equality (equals() method) to compare key and value. HashMap uses object equality (equals() method). A IdentityHashMap is typically used in scenarios where the keys are not unique based on object equality, but are unique based on their reference. Since == operator is faster than equals(), the IdentityHashMap is considered to be faster than the HashMap.

1. **What is the difference between an ArrayList and a LinkedList in Java Collections?**

An ArrayList is a collection in Java that stores elements in a dynamic array, while a LinkedList stores elements in a linked list of nodes. An ArrayList is more efficient for accessing elements by index, while a LinkedList is more efficient for adding or removing elements at the beginning or the end of the list.

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| To store elements, ArrayList internally uses a dynamic array. | To store elements, LinkedList internally uses a doubly linked list, where each element (node) holds a reference to the previous and next nodes. |
| As ArrayList internally uses an array, manipulation with ArrayList is slow. If we remove any element from the array, all the remaining elements get shifted in memory. | As LinkedList internally uses doubly linked list, manipulation with LinkedList is faster as compared to ArrayList. In this case, no bit shifting happens in memory. |
| Random access (accessing elements by index) is fast because elements are stored in a contiguous block of memory. | Random access is slower because it requires traversing the list from the beginning. |
| For storing and accessing data ArrayList is better. | For data manipulation LinkedList is better. |
| It generally has a higher memory overhead compared to LinkedList. In addition to storing the elements, it also allocates additional memory for potential growth. | It has a lower memory overhead compared to ArrayList because it only needs to store the elements and the references to the previous and next nodes. |
| Generally, At the time of initialization ArrayList has a default capacity of 10 elements. | The LinkedList doesn’t have a default capacity. An empty list is created when a LinkedList is initialized. |
| ArrayList provides fast random access, but slower insertions and deletions in the middle | LinkedList offers efficient insertions and deletions, but slower random access. |

1. **How is an ArrayList implemented in Java Collections?**

An ArrayList in Java Collections is implemented as a resizable array using the Object class array (Object[]) data structure. The implementation uses a capacity and size variable to locate the current capacity and the number of elements in the array. By default, ArrayList generates an array of size 10. While initializing the Array, we can specify the size of the array. When adding or removing elements, the space in the ArrayList automatically increases or decreases. The ArrayList class provides methods for adding, removing, and accessing elements by index.

1. **Explain the HashMap data structure and its usage?**

Suppose we have a requirement where there is a need to use the key and value in pair, we should use the Map interface in Java. HashMap is an implementation of the Map interface and provides a key-value pair partnership. It uses a hash table data structure to store and retrieve elements.

When we don’t need to maintain any particular order of elements and no restriction on null keys and values, we can use HashMap.

HashMap provides methods like put(key, value), get(key), and remove(key) to handle the key-value pairs. It also offers various other methods for iterating over the keys or values, checking if a key or value exists, and more.

1. **What is the difference between a HashSet and a TreeSet?**

A HashSet is a collection in Java that stores elements in a hash table using their hash codes. A TreeSet stores elements in a tree like structure in their natural ordering or a defined comparator. A HashSet is more efficient for adding, removing, and querying elements. On the other hand, a TreeSet is more efficient for finding the smallest or largest element in the set.

1. **How is a HashSet implemented?**

A HashSet in Java Collections is implemented as a hash table using a HashMap instance with a dummy value as the value for each key. The hash code of each element is computed using the hashCode() method, and the element is stored in the hash table based on its hash code. The implementation uses a load factor and capacity to manage the size of the hash table, and provides methods for adding, removing, and querying elements.

1. **What is the difference between a HashMap and a TreeMap?**

A HashMap stores key-value pairs in a hash table based on their hash codes, while a TreeMap stores key-value pairs in a tree structure based on their natural ordering or a specified comparator. A HashMap can add, remove, and query key-value pairs more efficiently, while a TreeMap is more efficient for finding key-value pairs within a defined range.

1. **How is a HashMap implemented in Java Collections?**

A HashMap in Java Collections is implemented as a hash table using an array of linked lists to handle hash code collisions. By chaining several key-value pairs in the same bucket, the linked list is utilized to handle collisions. Each key-value pair is stored in a bucket according to the hash code of the key. The approach offers methods for adding, removing, and querying key-value pairs and uses a load factor and capacity to regulate the size of the hash table.

1. **How does HashMap work in Java?**

HashMap in java utilizes it’s inner class Node<K,V> to store the mappings. HashMap internally works on hashing algorithm and uses hashCode() and equals() method on key for ‘get’ and ‘put’ operations. HashMap also uses singly linked list to save elements, which are called bins or buckets.

When we call put method, hashCode of key is used to find out the bucket to store the mapping. Once bucket is determined, hashCode is used to verify if there is already a key with same hashCode or not. If there is an existing key with same hashCode, then equals() method is used on the key. If equals() returns true, then value is overwritten, or else a new mapping is produced to this singly linked list bucket. If there is no key with same hashCode then mapping is added into the bucket.

For HashMap get operation, again key hashCode is used to find out the bucket to check for the value. After the bucket is determined, entries are traversed to find out the Entry using hashCode and equals() method. If a match is found, the value is returned, otherwise null is returned. There are many more things involved in between, such as the hashing algorithm to get the bucket for the key, rehashing of mappings etc. But for our working, just keep in mind that HashMap operations work on Key and well implementation of hashCode and equals() method is essential to ignore the unwanted behavior.

1. **What is the difference between a LinkedHashMap and a HashMap?**

A LinkedHashMap stores key-value pairs in a hash table based on their hash codes, while maintaining the order of insertion using a linked list. On the other hand, a HashMap stores key-value pairs in a hash table based on their hash codes, but does not maintain the order of insertion. A LinkedHashMap is more efficient for iteration in the order of insertion, while a HashMap is more efficient for adding, removing, and querying key-value pairs.

1. **How does a concurrentHashMap work in Java Collections?**

The ConcurrentHashMap in Java Collections is a thread-safe implementation of a hash table. It allows multiple threads to access and modify the map concurrently without the need for external synchronization. It uses a partitioned array of segments to ensure that different segments can be accessed and modified concurrently by different threads. Each segment contains a hash table based on the hash codes of the keys, and supports lock-free read and write operations to ensure high concurrency.

1. **What is the difference between a CopyOnWriteArrayList and an ArrayList?**

A CopyOnWriteArrayList is a thread-safe implementation that allows for efficient reads and concurrent modifications without the need for external synchronization. It creates a new copy of the array each time an element is added, removed or modified. In this way, it ensures that the original array remains unchanged and can be read by other threads safely. An ArrayList is not thread-safe by default, it requires external synchronization to ensure safe access and modification by multiple threads.

1. **What is the difference between a PriorityQueue and a Queue?**

As the name suggests, a PriorityQueue is an extension of Queue Collection that orders its elements based on their priority, while a Queue is a collection that orders its elements based on their order of insertion. A PriorityQueue is typically used in algorithms where elements need to be processed based on their priority, while a Queue is used in algorithms where elements need to be processed in the order they were added.

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

A LinkedList is a collection in Java that implements a doubly linked list data structure to store its elements. On the other hand, an ArrayList implements a dynamic array data structure. A LinkedList is more efficient in adding or removing elements from the beginning or end of the list, while an ArrayList is more efficient in random access and traversal of the elements in the list.

1. **What is the usage of a Stack and a Queue in Java Collections?**

Generally, both are used in implementing the algorithms. A Stack is typically used where elements need to be processed in reverse order, while a Queue is used where elements need to be processed in the order they were added.

1. **What is the difference between a TreeMap and a HashMap in Java Collections?**

A TreeMap is a collection in Java that stores elements in a tree data structure based on their keys. On the other hand, a HashMap stores elements in a hash table based on their hash codes. A TreeMap maintains the elements in sorted order based on their keys, while a HashMap does not guarantee any particular order.

1. **What is the difference between a WeakHashMap and a HashMap in Java Collections?**

A HashMap uses strong references to keys, whereas a WeakHashMap uses weak references. WeakHashMap allows the garbage collector to free up memory when the key is no longer referenced. When it is advisable to let the garbage collector release memory when the key is no longer required, a WeakHashMap is often used.

1. **What is the difference between a ConcurrentHashMap and a HashMap in Java Collections?**

A ConcurrentHashMap is a thread-safe implementation of a HashMap. It is particularly intended for use in multi-threaded and concurrent environment. In contrast, a HashMap is not thread-safe, and requires external synchronization to work in multi-threaded and concurrent environment.

1. **What is the difference between a EnumSet and a HashSet?**

An EnumSet is a collection in Java that stores elements of a defined enum type as bit vectors. A HashSet stores elements in a hash table based on their hash codes. An EnumSet is normally used to work with enums, while a HashSet can be used with any type of object. An EumSet works faster than the HashSet as it is a high performing set implementation. Also, EnumSet doesn’t compute any hashCode to find the right bucket.