6

Q1. What is Collection in Java?
In Java, the term "Collection" refers to a framework that provides a set of classes and interfaces to store, manipulate, and process groups of objects. It is part of the Java Collections Framework, which is built into the Java API and provides ready-to-use data structures and algorithms for handling collections of objects.
The Collection framework in Java includes interfaces, such as List, Set, Queue

Q2. Differentiate between Collection and collections in the context of Java.

- Collection: In Java, the term "Collection" refers to the entire framework that provides interfaces and classes to handle groups of objects. It represents a single unit that holds multiple elements.
- collections: The term "collections" (lowercase) is a general term that refers to multiple instances or objects of the Collection classes. It represents multiple collections or a collection of collections.

Q3. What are the advantages of the Collection framework?
The advantages of the Collection framework in Java are:

- Reusability: The Collection framework provides a set of reusable data structures and algorithms that can be used to handle collections of objects in a generic way. It eliminates the need to write custom data structures and algorithms for common collection operations.
- Interoperability: The Collection framework provides a common set of interfaces and methods, allowing collections to be easily interchanged and used together. This promotes code reusability and enhances modularity.
- Efficiency: The Collection framework offers efficient data structures and algorithms for various collection types, such as arrays, lists, sets, queues, and maps. These data structures are designed to optimize storage, retrieval, and manipulation operations.

Q4. Explain the various interfaces used in the Collection framework.
The Collection framework in Java includes several interfaces, such as:

- Collection: It is the root interface that represents a group of objects known as a collection. It defines basic operations like adding, removing, and iterating over elements.
- List: It is an ordered collection that allows duplicate elements. It provides operations for positional access, such as getting an element by index, adding elements at specific positions, and removing elements by index.
- Set: It is an unordered collection that does not allow duplicate elements. It provides methods for adding, removing, and checking the presence of elements in the set.
- Queue: It is a collection designed for holding elements prior to processing. It follows the FIFO (First-In-First-Out) ordering principle, where elements are inserted at the end and removed from the beginning.
- Map: It is an object that maps keys to values. It does not inherit from the Collection interface but is part of the Collection framework. It provides methods to associate keys with values, retrieve values by keys, and perform operations based on key-value pairs.

Q5. Differentiate between List and Set in Java.

- List: List is an ordered collection that allows duplicate elements. It preserves the insertion order of elements, which means the elements are stored in the same order as they are added to the list. Lists provide positional access, meaning elements can be accessed by their index. Examples of List implementations include ArrayList, LinkedList, and Vector.
- Set: Set is an unordered collection that does not allow duplicate elements. It does not maintain any specific order of elements. Set implementations guarantee uniqueness of elements by using the equals() and hashCode() methods. Examples of Set implementations include HashSet, TreeSet, and LinkedHashSet

Q6. What is the difference between Iterator and ListIterator in Java?

- Iterator: The Iterator interface provides a way to iterate over a collection in a forward direction. It allows sequential access to the elements of a collection and provides methods like hasNext() to check if there are more elements, and next() to retrieve the next element in the iteration. The Iterator interface is available in the java.util package.
- ListIterator: The ListIterator interface is a sub-interface of Iterator and provides additional functionalities specifically for iterating over lists. It extends the Iterator interface and allows bidirectional traversal of elements. In addition to the methods provided by Iterator, ListIterator also provides methods like hasPrevious(), previous(), nextIndex(), previousIndex(), and set() for more advanced list traversal and modification operations.

Q7. What is the difference between Comparable and Comparator?

- Comparable: Comparable is an interface in Java that is used to define the natural ordering of objects. It allows objects of a class to be compared and sorted based on their own implementation of the compareTo() method. The compareTo() method compares the current object with another object and returns a negative integer, zero, or a positive integer depending on whether the current object is less than, equal to, or greater than the other object. The Comparable interface is typically implemented by the class whose objects are being compared.
- Comparator: Comparator is an interface in Java that is used to define custom comparison logic for objects that do not implement the Comparable interface or when a different sorting order is required. It provides a way to compare objects based on specific criteria defined by the programmer. The Comparator interface contains a single method called compare(), which takes two objects as arguments and returns a negative integer, zero, or a positive integer based on the comparison logic. The compare() method is implemented separately from the objects being compared.

Q8. What is collision in HashMap?

In Java, HashMap is a hash table-based implementation of the Map interface. A collision in a HashMap occurs when two or more keys map to the same bucket or hash value. This can happen due to the nature of the hashing algorithm and the limited number of buckets in the HashMap.

When a collision occurs, the HashMap uses a linked list or a balanced tree (in Java 8 and later versions) to store multiple entries with the same hash value in the same bucket. Each entry consists of a key-value pair. The linked list or tree allows efficient retrieval and storage of entries with the same hash value.

To retrieve a value based on a key, the HashMap computes the hash code of the key, determines the bucket where the entry should be located, and then iterates over the linked list or tree (if present) to find the exact match based on the key's equals() method.

Q9. Distinguish between a HashMap and a TreeMap.

- HashMap: HashMap is an implementation of the Map interface in Java that provides key-value mappings. It uses hashing techniques to store and retrieve entries based on their hash codes. HashMap provides constant-time performance for basic operations like get() and put().
- TreeMap: TreeMap is another implementation of the Map interface that stores key-value pairs in a sorted order based on the natural ordering of the keys or a custom comparator. It uses a Red-Black tree as the underlying data structure. TreeMap provides sorted key-value mappings and allows efficient range searches and ordered iteration.

The main differences between HashMap and TreeMap are:

- Ordering: HashMap does not maintain any particular order of the elements, while TreeMap orders the elements based on the natural ordering of the keys or a custom comparator.
- Performance: HashMap provides constant-time performance for basic operations like get() and put(), while TreeMap provides O(log N) time complexity for these operations due to the underlying Red-Black tree

structure.

- Null Keys: HashMap allows a single null key, while TreeMap does not allow null keys as it relies on the natural ordering or custom comparator for sorting.

Q10. Define LinkedHashMap in Java.

LinkedHashMap is an implementation of the Map interface in Java that extends the HashMap class. It maintains a doubly-linked list of entries in addition to the hash table, which allows predictable iteration order based on the insertion order of elements. Each entry in a LinkedHashMap maintains a reference to the next and previous entries in the insertion order.

The key characteristics of LinkedHashMap are:

- Iteration Order: LinkedHashMap guarantees that the iteration order of the entries will be the same as the order in which they were inserted into the map. This makes it useful in scenarios where the order of insertion needs to be preserved.
- Performance: The performance of LinkedHashMap is generally slightly lower than that of HashMap due to the additional bookkeeping of maintaining the linked list. However, the difference in performance is usually negligible for most use cases.
- Null Keys and Values: Similar to HashMap, LinkedHashMap allows a single null key and multiple null values.