**Assignment Questions 6**

<aside> 💡 **Q1.What is Collection in Java?**

</aside>

<aside> 💡 **Q2. Differentiate between Collection and collections in the context of Java.**

</aside>

<aside> 💡 **Q3. What are the advantages of the Collection framework?**

</aside>

<aside> 💡 **Q4.Explain the various interfaces used in the Collection framework.**

</aside>

<aside> 💡 **Q5.Differentiate between List and Set in Java.**

</aside>

<aside> 💡 **Q6.What is the Differentiate between Iterator and ListIterator in Java.**

</aside>

<aside> 💡 **Q7.What is the Differentiate between Comparable and Comparator**

</aside>

<aside> 💡 **Q8.What is collision in HashMap?**

</aside>

<aside> 💡 **Q9.Distinguish between a hashmap and a Treemap.**

</aside>

<aside> 💡 **Q10.Define LinkedHashMap in Java**

</aside>

Q1. What is a Collection in Java?

In Java, a Collection is an interface that represents a group of objects, known as elements. It provides a unified and standardized way of storing, manipulating, and accessing a group of objects. The Collection interface is a part of the Java Collections Framework and is implemented by various classes such as List, Set, and Queue.

Collections in Java provide functionality for adding, removing, and retrieving elements, as well as performing operations such as searching, sorting, and iterating over the elements. They offer flexibility in terms of the size of the collection and the types of objects that can be stored.

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

* Collection (with a capital 'C') refers to the Collection interface in Java's Collections Framework. It is an interface that represents a group of objects and provides operations for manipulating and accessing those objects. Classes such as List, Set, and Queue implement this interface.
* collections (with a lowercase 'c') refer to the general concept of multiple objects grouped together. It is a more generic term that can refer to any type of collection, including arrays or custom data structures.

In summary, Collection is a specific interface in Java's Collections Framework, while collections is a broader term referring to any grouping of objects.

Q3. What are the advantages of the Collection framework?

The Collection framework in Java provides several advantages:

1. Reusability: The Collection framework provides reusable data structures and algorithms for storing, manipulating, and accessing groups of objects. It eliminates the need to implement these functionalities from scratch, saving development time and effort.
2. Standardization: The Collection framework defines a set of common interfaces, such as Collection, List, Set, and Map, along with their corresponding implementations. This standardization allows different parts of an application to interact seamlessly and promotes code interoperability.
3. Extensibility: The Collection framework allows for easy extension and customization. It provides interfaces and abstract classes that can be extended and implemented to create custom collection classes with specific behaviors or requirements.
4. Performance: The Collection framework provides efficient implementations of data structures, such as ArrayList, LinkedList, HashSet, and TreeMap, optimized for different use cases. These implementations are backed by well-established algorithms and data structures, resulting in good performance characteristics.
5. Increased Productivity: The Collection framework simplifies common collection-related tasks, such as sorting, searching, and iteration, by providing built-in methods and utilities. This improves productivity by reducing the need to write complex and error-prone code.
6. Compatibility: The Collection framework is designed to be compatible with existing Java libraries and APIs. It integrates well with other Java features, such as generics, streams, and lambda expressions, making it easier to work with collections in modern Java applications.

Q4. Explain the various interfaces used in the Collection framework.

The Collection framework in Java includes several interfaces that define the contracts and behaviors of different types of collections. Some of the key interfaces used in the Collection framework are:

* Collection: It is the root interface that defines the basic operations common to all collections, such as adding, removing, and iterating over elements. It extends the Iterable interface.
* List: It is an ordered collection that allows duplicate elements. It maintains the insertion order of elements and provides positional access to elements. Common implementations include ArrayList and LinkedList.
* Set: It is an unordered collection that does not allow duplicate elements. It enforces uniqueness of elements based on the equals() method. Common implementations include HashSet and TreeSet.
* Queue: It is a collection designed for holding elements prior to processing. It follows the FIFO (First-In-First-Out) or LIFO (Last-In-First-Out) ordering. Common implementations include LinkedList and PriorityQueue.
* Map: It is an object that maps keys to values. It does not extend the Collection interface but is an important part of the Collection framework. It provides methods for storing, retrieving, and manipulating key-value pairs. Common implementations include HashMap and TreeMap.
* Iterator: It is an interface that provides methods for iterating over the elements of a collection in a forward direction. It allows sequential access to elements.
* ListIterator: It is a sub-interface of Iterator that provides additional methods for bidirectional iteration, allowing traversal in both forward and backward directions.

These interfaces, along with their implementing classes, provide a wide range of options for storing and manipulating collections of objects in Java.

Q5. Differentiate between List and Set in Java.

List and Set are both interfaces in the Java Collections Framework that represent different types of collections. Here are the differences between List and Set:

* List:
  + Allows duplicate elements: List implementations allow duplicate elements, meaning you can store multiple objects with the same value.
  + Maintains insertion order: List implementations maintain the order in which elements are inserted. Elements are accessed based on their index.
  + Access by index: Lists provide positional access to elements. You can retrieve elements by their index using methods like **get()** and modify elements using methods like **set()**.
  + Common implementations: ArrayList and LinkedList are common implementations of the List interface.
* Set:
  + Does not allow duplicate elements: Set implementations do not allow duplicate elements. Each element in a set must be unique based on its equals() method.
  + No defined order: Set implementations do not define a specific order for elements. The order in which elements are stored or retrieved may vary.
  + Fast membership check: Set implementations provide fast membership checks to determine whether an element is present or not.
  + Common implementations: HashSet and TreeSet are common implementations of the Set interface.

In summary, List allows duplicates, maintains insertion order, and provides access by index. Set does not allow duplicates, does not maintain a specific order, and provides fast membership checks. The choice between List and Set depends on the specific requirements of your application.

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

Iterator and ListIterator are interfaces in Java used for traversing collections. Here are the differences between them:

* Iterator:
  + Supports forward iteration only: The Iterator interface provides methods for iterating over elements in a collection in a forward direction. It allows sequential access to elements.
  + Available for all collection types: The Iterator interface is available for all collection types, including List, Set, and Queue.
  + Limited functionality: It provides basic operations such as **next()**, **hasNext()**, and **remove()**. It does not support adding or modifying elements during iteration.
* ListIterator:
  + Supports bidirectional iteration: The ListIterator interface is a sub-interface of Iterator that provides additional methods for bidirectional iteration. It allows traversal in both forward and backward directions.
  + Available only for List implementations: The ListIterator interface is available only for List implementations, as it depends on the underlying list's indexing capabilities.
  + Extended functionality: In addition to the methods provided by Iterator, ListIterator supports operations like **previous()**, **hasPrevious()**, **add()**, and **set()**. It allows modification and insertion of elements during iteration.

In summary, Iterator supports forward iteration and is available for all collection types, while ListIterator supports bidirectional iteration and is specifically designed for List implementations. ListIterator provides extended functionality compared to Iterator, allowing modification and insertion of elements during iteration.

Q7. What is the difference between Comparable and Comparator?

Comparable and Comparator are interfaces in Java used for object comparison. Here are the differences between them:

* Comparable:
  + Defined in the object's class: The Comparable interface is implemented by the objects themselvesto define their natural ordering. It is typically used for objects that have an inherent natural order, such as numbers or strings.
  + compareTo() method: The Comparable interface defines a single method, **compareTo()**, which is used to compare an object with another object. It returns a negative integer, zero, or a positive integer based on the comparison result.
* Comparator:
  + Defined separately from the object's class: The Comparator interface is implemented by separate classes to define custom comparison logic for objects. It allows for multiple ways of comparing objects, depending on the requirements.
  + compare() method: The Comparator interface defines a method, **compare()**, which takes two objects as arguments and returns a negative integer, zero, or a positive integer based on the comparison result. This allows for custom comparison logic that is different from the object's natural ordering.
  + Flexibility: Comparators provide flexibility in choosing different comparison criteria or strategies for objects. They can be used to sort objects in various ways without modifying their original class implementation.

In summary, Comparable is implemented by the objects themselves to define their natural ordering, while Comparator is implemented by separate classes to define custom comparison logic for objects. Comparable uses the **compareTo()** method for comparison, while Comparator uses the **compare()** method. Comparators provide flexibility in choosing different comparison criteria or strategies for objects.