15. Collections Framework

1) Introduction to Collections Framework

Answer:

The **Collections Framework** in Java provides a set of classes and interfaces that implement commonly used data structures such as lists, sets, queues, and maps. It provides a unified architecture for handling collections of objects, enabling developers to work with data in a flexible and efficient way.

The Collections Framework is part of the **java.util** package and consists of several interfaces, implementations, and algorithms. It simplifies the development of programs by providing data structures that are easy to use and manage.

2) List, Set, Map, and Queue Interfaces

Answer:

The **Collections Framework** in Java provides several core interfaces that help in storing and manipulating data efficiently. These interfaces can be broadly categorized into four groups: **List**, **Set**, **Map**, and **Queue**. Each of these interfaces has specific characteristics and is implemented by different classes that provide different behaviors.

3) ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap

Answer:

1. ArrayList

- **Definition**: ArrayList is a resizable array implementation of the List interface. It allows dynamic resizing, meaning it can grow and shrink as needed when elements are added or removed.
- **Characteristics**:
    - Allows **duplicates**.
    - Maintains **insertion order**.
    - **Indexed access** to elements (constant time complexity for access by index).
    - **Not synchronized** (not thread-safe).
    - **Fast** for random access but slower for insertions/removals in the middle.
- **Common Methods**:
    - add(E e): Adds an element to the list.

    - get(int index): Retrieves an element at a specified index.
    - remove(int index): Removes an element at a specified index.
    - contains(Object o): Checks if an element is in the list.
    - size(): Returns the number of elements in the list.

2. LinkedList

- **Definition**: LinkedList is a doubly linked list implementation of the List interface. It allows elements to be inserted or removed at both ends (efficient for queue and stack operations).
- **Characteristics**:
    - Allows **duplicates**.
    - Maintains **insertion order**.
    - Provides **efficient insertions/removals** at the beginning or end of the list, but **slower for random access**.
    - Can be used as a **Queue** (FIFO) or **Deque** (double-ended queue).
- **Common Methods**:
    - add(E e): Adds an element to the list.
    - get(int index): Retrieves an element at a specified index.
    - remove(int index): Removes an element at a specified index.
    - offerFirst(E e): Adds an element at the beginning.
    - offerLast(E e): Adds an element at the end.
    - pollFirst(): Retrieves and removes the first element.

3. HashSet

- **Definition**: HashSet is an implementation of the Set interface that uses a hash table for storage. It does not allow duplicate elements and does not guarantee any specific order of elements.
- **Characteristics**:
    - **Does not allow duplicates**.
    - Does not maintain **insertion order**.
    - Provides **constant time complexity** for basic operations like add(), remove(), and contains().
    - **Not synchronized** (not thread-safe).
- **Common Methods**:
    - add(E e): Adds an element to the set.
    - remove(Object o): Removes an element from the set.
    - contains(Object o): Checks if an element is in the set.
    - size(): Returns the number of elements in the set.

4. TreeSet

- **Definition**: TreeSet is a Set implementation based on a Red-Black tree. It stores elements in **sorted order**, either by their natural ordering or by a comparator provided at the time of creation.

- **Characteristics**:
  - **Does not allow duplicates**.
  - **Sorted** according to natural ordering or by a comparator.
  - Slower than HashSet for basic operations but provides **sorted access**.
  - **Not synchronized** (not thread-safe).
- **Common Methods**:
  - add(E e): Adds an element to the set.
  - remove(Object o): Removes an element from the set.
  - contains(Object o): Checks if an element is in the set.
  - size(): Returns the number of elements in the set.

5. HashMap

- **Definition**: HashMap is an implementation of the Map interface that uses a hash table for storage. It stores key-value pairs, where the keys must be unique.
- **Characteristics**:
  - **Allows one null key** and **multiple null values**.
  - **Does not maintain order** of the elements (keys are unordered).
  - Provides **constant-time complexity** for get() and put() operations, assuming no hash collisions.
  - **Not synchronized** (not thread-safe).
- **Common Methods**:
  - put(K key, V value): Adds a key-value pair to the map.
  - get(Object key): Retrieves the value for the given key.
  - remove(Object key): Removes the key-value pair.
  - containsKey(Object key): Checks if the map contains a given key.
  - size(): Returns the number of key-value pairs in the map.

6. TreeMap

- **Definition**: TreeMap is a Map implementation that is based on a Red-Black tree and stores the key-value pairs in **sorted order** according to the natural ordering of the keys or a custom comparator.
- **Characteristics**:
  - **Does not allow null keys** but allows null values.
  - **Sorted by key**, either by their natural ordering or by a comparator.
  - Slower than HashMap for basic operations due to the sorting requirement.
  - **Not synchronized** (not thread-safe).
- **Common Methods**:
  - put(K key, V value): Adds a key-value pair to the map.
  - get(Object key): Retrieves the value for the given key.
  - remove(Object key): Removes the key-value pair.
  - firstKey(): Returns the first (lowest) key in the map.
  - lastKey(): Returns the last (highest) key in the map.

4) Iterators and ListIterators

Answer:

1. Iterator

The **Iterator** interface provides methods for iterating over the elements of a collection, such as List, Set, or Queue. It allows you to traverse the collection, check for elements, and remove elements during iteration.

2. ListIterator

The **ListIterator** is a more powerful version of the Iterator interface. It is specific to the List interface (i.e., it can only be used with List collections such as ArrayList and LinkedList). ListIterator extends Iterator and provides additional methods for bidirectional traversal and element modification.

==========
==========
==========
==========