**Assignment: (Core Java)**

**15: Collections Framework**

**Que.1 Introduction to Collections Framework**

**Ans.1** A collection in Java is a group of individual objects that are treated as a single unit. In Java, a separate framework named the "Collection Framework" was defined in JDK 1.2, which contains all the Java Collection Classes and interfaces. In Java, the Collection interface and Map interface are the two main “root” interfaces of Java collection classes.

Before the Collection Framework, Java used Arrays, Vectors, and Hashtables to group objects, but they lacked a common interface. Each had a separate implementation, making usage inconsistent and harder for developers to learn and maintain.

**Advantages of the Java Collection Framework:** Since the lack of a collection framework gave rise to the above set of disadvantages, the following are the advantages of the collection framework.

1. **Consistent API:**Interfaces like List, Set, and Map have common methods across classes**.**
2. **Less Coding Effort:** Developers focus on usage, not designing data structures—supports OOP abstraction.
3. **Better Performance:** Offers fast, reliable implementations of data structures, improving speed and quality of code.

**Que.2 List, Set, Map, and Queue Interfaces**

**Ans.2** All collection classes are based on these core interfaces:

1. Collection (root interface)
   * Subinterfaces: List, Set, Queue
2. List
   * Ordered, allows duplicates
   * Implementations: ArrayList, LinkedList, Vector, Stack
3. Set
   * No duplicates allowed
   * Implementations: HashSet, LinkedHashSet, TreeSet
4. Queue
   * FIFO (First In First Out), some are priority-based
   * Implementations: PriorityQueue, ArrayDeque
5. Map (not child of Collection)
   * Stores key–value pairs (unique keys)
   * Implementations: HashMap, LinkedHashMap, TreeMap, Hashtable

**Iterable**

**Collection**

**List Set Queue**

**SortedSet**

**Map (separate hierarchy)**

**Que.3 ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap**

**Ans.3 1. ArrayList**

* Implements: List
* Features:
  + Dynamic array
  + Maintains insertion order
  + Allows duplicates and null values
  + Random access is fast (index-based)
  + Slower insert/delete in middle

**2. LinkedList**

* Implements: List, Deque
* Features:
  + Doubly-linked list
  + Maintains insertion order
  + Allows duplicates and null values
  + Faster insertion/deletion compared to ArrayList
  + Slower random access

**3. HashSet**

* Implements: Set
* Features:
  + Uses HashMap internally
  + No duplicates allowed
  + At most one null element
  + Does not guarantee order
  + Fast lookups (hashing)

**4. TreeSet**

* Implements: NavigableSet (SortedSet)
* Features:
  + Stores elements in sorted order
  + No duplicates allowed
  + Does not allow null
  + Slower than HashSet

**5. HashMap**

* Implements: Map
* Features:
  + Stores key–value pairs
  + Keys: unique, one null allowed
  + Values: can be duplicates, multiple nulls allowed
  + No order guarantee
  + Fast lookups (hashing)

**6. TreeMap**

* Implements: NavigableMap (SortedMap)
* Features:
  + Stores key–value pairs in sorted order of keys
  + Keys: unique, null key not allowed
  + Values: can be duplicate
  + Slower than HashMap

**Que.4 Iterators and ListIterators**

**Ans.4** In Java, both Iterator and ListIterator are interfaces used for traversing elements within collections, but they differ in their capabilities and the types of collections they can operate on.

Iterator:

* The Iterator interface provides a universal way to iterate over elements in any class that implements the Collection interface .
* It supports only forward iteration, meaning elements can be traversed from the beginning to the end.
* Key methods include:
  + hasNext(): Checks if there are more elements to iterate.
  + next(): Returns the next element in the iteration.
  + remove(): Removes the last element returned by next() from the underlying collection.

ListIterator:

* The ListIterator interface is a sub-interface of Iterator and is specifically designed for traversing elements in classes that implement the List interface.
* It supports bi-directional iteration, allowing traversal both forward and backward.
* It offers more advanced functionalities compared to Iterator, including:
  + **Bi-directional traversal:** hasPrevious(), previous().
  + **Index access:** nextIndex(), previousIndex().
  + **Element modification:** add(E e) (inserts an element), set(E e) (replaces the last element returned).