**Core Java - Assignment**

**Module - 1**

1. **Collections Framework :**

**Theory : -**

1. Introduction to Collections Framework:

-> The Java Collections Framework is a unified architecture for representing and manipulating collections of objects. It provides a set of interfaces and classes that implement various data structures and algorithms for storing, retrieving, and manipulating collections of objects. The framework is part of the java. util package and includes interfaces, implementations, and algorithms.

-> Interfaces: Abstract data types that represent collections. The primary interfaces are Collection, List, Set, Queue, and Map.

-> Implementations: Concrete classes that implement the collection interfaces. Examples include ArrayList, LinkedList, HashSet , TreeSet, HashMap, and Treemap .

-> Algorithms: Methods that perform useful computations, such as searching and sorting, on objects that implement collection interfaces. These are usually provided as static methods in the Collections utility class.

-> Key Interfaces

-> Collection: The root interface for all collection classes.

-> List: An ordered collection (also known as a sequence) that allows duplicate elements.

-> Set: A collection that does not allow duplicate elements.

-> Queue: A collection used to hold multiple elements prior to processing.

-> Map: An object that maps keys to values, with no duplicate keys allowed.

-> Key Classes

-> ArrayList: A resizable array implementation of the List interface.

-> LinkedList: A doubly-linked list implementation of the List and Deque interfaces.

-> HashSet: A hash table implementation of the Set interface.

-> TreeSet A sorted tree implementation of the Set interface.

-> HashMap: A hash table implementation of the Map interface.

-> TreeMap: A sorted tree implementation of the Map interface.

-> PriorityQueue: A priority heap implementation of the Queue interface.

1. List, Set, Map, and Queue Interfaces :

-> List Interface : The List interface represents an ordered collection (also known as a sequence) that allows duplicate elements. Elements can be accessed by their integer index, and the order of elements is maintained.

->Key Methods :

-> add(E e) : Appends the specified element to the end of the list.

-> get(int index) : Returns the element at the specified position in the list.

-> remove(int index) : Removes the element at the specified position in the list.

-> size() : Returns the number of elements in the list.

-> contains (Object o) : Returns true if the list contains the specified element.

-> Set Interface : The Set interface represents a collection that does not allow duplicate elements. It models the mathematical set abstraction and is typically used to store unique elements.

-> Key Methods

-> add(E e) : Adds the specified element to the set if it is not already present.

-> remove(Object o) : Removes the specified element from the set if it is present.

-> size() : Returns the number of elements in the set.

-> contains (Object o) : Returns true if the set contains the specified element.

-> Map Interface : The map interface represents a collection of key-value pairs. Each key can map to at most one value, and duplicate keys are not allowed.

-> Key Methods

-> put(K key, V value) : Associates the specified value with the specified key in the map.

-> get(Object key) : Returns the value to which the specified key is mapped, or null if the map contains no mapping for the key.

-> remove(Object key) : Removes the mapping for the specified key from the map if present.

-> size() : Returns the number of key-value mappings in the map.

-> containsKey(Object key) : Returns true if the map contains a mapping for the specified key.

-> Queue Interface : The Queue interface represents a collection designed for holding elements prior to processing. It typically orders elements in a FIFO (first-in-first-out) manner.

-> Key Method :

-> add(E e) : Inserts the specified element into the queue.

-> remove() : Retrieves and removes the head of the queue.

-> peek() : Retrieves, but does not remove, the head of the queue.

-> poll() : Retrieves and removes the head of the queue, or returns null if the queue is empty.

-> size() : Returns the number of elements in the queue.

1. ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap

-> ArrayList : ArrayList is a resizable array implementation of the List interface. It allows random access to elements and is efficient for retrieving elements by index.

-> LinkedList : LinkedList is a doubly-linked list implementation of the List and Deque interfaces. It is efficient for inserting and removing elements at both ends of the list.

-> HashSet : HashSet is a hash table implementation of the Set interface. It does not allow duplicate elements and does not maintain any order of elements.

-> TreeSet : TreeSet is a sorted tree implementation of the Set interface. It does not allow duplicate elements and maintains elements in a sorted order.

-> HashMap : HashMap is a hash table implementation of the map interface. It allows null values and one null key, and does not maintain any order of elements.

-> TreeMap : Treemap is a sorted tree implementation of the Map interface. It maintains elements in a sorted order based on the natural ordering of the keys or by a comparator provided at map creation time.

1. Iterators and Listlterators

-> Iterator Interface : The Iterator interface provides methods to iterate over a collection. It is a universal iterator that can be used with any collection that implements the Collection interface.

-> Key Methods

-> boolean hasNext() : Returns true if the iteration has more elements.

-> E next() : Returns the next element in the iteration.

-> Listlterator Interface : The Listlterator interface extends the Iterator interface and provides additional methods to traverse a list in both directions (forward and backward) and to modify the list during iteration. It is only available for lists.

-> Key Methods

-> hasNext() : Returns true if the iteration has more elements when traversing forward.

-> next() : Returns the next element in the iteration.

-> hasPrevious() : Returns true if the iteration has more elements when traversing backward.

-> previous() : Returns the previous element in the iteration.

-> nextlndex() : Returns the index of the element that would be returned by a subsequent call to next() .

-> previouslndex() : Returns the index of the element that would be returned by a subsequent call to previous() .

-> set(E e) : Replaces the last element returned by the iterator with the specified element.

-> add(E e) : Inserts the specified element into the list.