**COLLECTIONS**

* The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.
* Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
* Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes ([ArrayList](https://www.javatpoint.com/java-arraylist), Vector,L[inkedList](https://www.javatpoint.com/java-linkedlist), [PriorityQueue](https://www.javatpoint.com/java-priorityqueue), HashSet, LinkedHashSet, TreeSet).

**Core Interfaces:**

* **Collection:** The root interface of the collections hierarchy. It represents a group of objects known as elements. Subinterfaces include List, Set, and Queue.
* **List:** An ordered collection (sequence) that allows duplicate elements. Implementations include ArrayList, LinkedList, and Vector.
* **Set:** A collection that contains no duplicate elements. Implementations include HashSet, TreeSet, and LinkedHashSet.
* **Queue:** A collection designed for holding elements prior to processing. Implementations include PriorityQueue and LinkedList.
* **Map:** An object that maps keys to values. It cannot contain duplicate keys, and each key can map to at most one value. Implementations include HashMap, TreeMap, and LinkedHashMap.

**Implementations:**

* **ArrayList:** A resizable array implementation of the List interface. Provides fast random access and dynamic resizing.
* **LinkedList:** A doubly-linked list implementation of the List interface. Provides fast insertion and deletion operations.
* **HashSet:** A hash table implementation of the Set interface. Provides constant-time performance for basic operations.
* **TreeSet:** A NavigableSet implementation backed by a TreeMap. Maintains elements in sorted order.
* **HashMap:** A hash table-based implementation of the Map interface. Provides constant-time performance for basic operations.
* **TreeMap:** A Red-Black tree-based implementation of the Map interface. Maintains key-value pairs in sorted order.

**Algorithms:**

The Collections class provides various static methods for performing common operations on collections, such as sorting, searching, and shuffling. Some notable methods include:

* **sort(Collection<T> c):** Sorts the elements of the specified collection in ascending order.
* **binarySearch(List<? extends Comparable<? super T>> list, T key):** Searches the specified list for the specified object using the binary search algorithm.
* **shuffle(List<?> list):** Randomly permutes the specified list using a default source of randomness.

**Thread Safety:**

Most collection implementations are not synchronized by default. However, thread-safe versions of collections can be obtained using the **Collections.synchronizedCollection()** method or using concurrent collection classes in the **java.util.concurrent** package.

**Use Cases:**

* **List:** Use when elements need to be accessed by index or when duplicates are allowed.
* **Set:** Use when uniqueness of elements is required.
* **Map:** Use when key-value pairs need to be stored and retrieved efficiently.
* **Queue:** Use when elements need to be processed in a specific order, such as FIFO (First-In-First-Out) or priority-based order.