# **Java Collection Framework**

**Java collection Framework:**

Java collection framework is a set of classes and interfaces that provide certain methods for implimenting various operations related tso that by creating its object.

**Collection:**

Collection is root class of all the interfaces (by using collection we are making our code flexable that we can change type of interface we want)

It does include the sub-interfaces (collection, List, Map, queue, dequeue, set).

Classes that implements the interface(ArrayList, LinkedList, TreeSet, Priorityqueue, Linkedhashedset, Arraydeque).

**Collection(interface) 🡪 subinterface 🡪 classes(objects)**

**we use arraylist or list ?**

**KeyPoints(ordered output,accesing with index)**

Using `ArrayList` directly instead of `List` (interface) ties your code to a specific implementation, limiting flexibility in switching to other list implementations like `LinkedList`. It also makes it harder to adhere to coding best practices that recommend coding to interfaces for improved maintainability and flexibility.

Using `List` (interface) instead of `ArrayList` allows for coding to an abstraction rather than a concrete implementation, promoting flexibility to easily switch between different list implementations (`ArrayList`, `LinkedList`, etc.) without changing much code. It also adheres to best practices in object-oriented design, enhancing code maintainability and scalability.

**Arrraylist and linked list are both are implimented from list interface both the diffrence between both is :**

### **ArrayList**

 It is used to store elements dynamically in a resizable array.

 Slow for insertion or deletion processes involving elements with an index because it may require shifting elements.

###  Fast in searching (random access) because accessing elements by index is O(1).

### **Dynamic Resizing**: When the array becomes full, ArrayList increases its size by creating a new array and copying the elements, which can be an expensive operation.

### **Iteration**: Iterating through an ArrayList is generally faster due to better cache locality because elements are stored contiguously in memory.

### **Memory Overhead:** Higher due to additional storage for node references.

**Use Cases**: Ideal for frequent additions and removals from the middle of the list.

**LinkedList :**

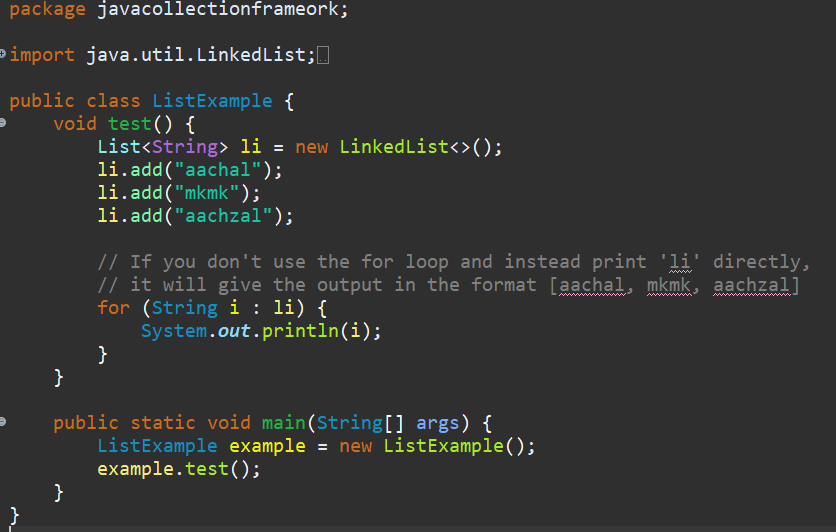
 In a linked list, nodes are connected to each other, where each node contains references to the previous and next nodes.

 Slow for searching because accessing elements by index requires traversing the list, which is O(n).

 Fast for insertion and deletion operations, especially when the location is known, because it only involves changing node references.

 **Memory Overhead**: Higher due to additional storage for node references.

 **Use Cases**: Ideal for frequent additions and removals from the middle of the list.



Output

Aachal mkmk aachazal

**Map**

### **HashMap<K, V>**

* **Characteristics**:
  + Implements Map using a hash table.
  + Allows null keys and values.
  + Unordered, meaning it does not guarantee any order of iteration.
  + Average time complexity for get and put operations is O(1).

### **2. LinkedHashMap<K, V>**

* **Characteristics**:
  + Extends HashMap and maintains a doubly-linked list running through all of its entries.
  + Preserves the order of insertion (insertion-order iteration).
  + Allows null keys and values.
  + Slightly slower than HashMap due to the overhead of maintaining the linked list.

### **3. TreeMap<K, V>**

* **Characteristics**:
  + Implements Map using a Red-Black tree.
  + Sorted according to the natural ordering of its keys or by a comparator provided at map creation time.
  + Does not allow null keys but allows null values.
  + Time complexity for get, put, and remove operations is O(log n).

### **4. Hashtable<K, V>**

* **Characteristics**:
  + Implements Map using a hash table.
  + Does not allow null keys or values.
  + Synchronized, meaning it is thread-safe.
  + Generally slower than HashMap due to synchronization overhead.