### **1. What is the Collection Framework in Java?**

The **Collection Framework** in Java is a set of classes and interfaces used to store, manage, and manipulate a group of objects (collections).

* It provides data structures like **List, Set, Queue, and Map** for efficient data management.
* Interfaces: List, Set, Queue, Map.
* Classes: ArrayList, LinkedList, HashSet, TreeSet, HashMap, etc.

**Key Advantages:**

* Reduces programming effort with ready-to-use data structures.
* Improves performance through efficient algorithms.
* Provides thread-safe collections with classes like ConcurrentHashMap.

### **2. What is the difference between ArrayList and LinkedList?**

| **Feature** | **ArrayList** | **LinkedList** |
| --- | --- | --- |
| **Implementation** | Backed by a **dynamic array**. | Backed by a **doubly linked list**. |
| **Access** | Faster for random access (index-based). | Slower for random access. |
| **Insertion/Deletion** | Slower, as elements need to be shifted. | Faster for frequent insertions/deletions. |
| **Memory Usage** | Uses less memory. | Uses more memory due to node overhead. |
| **Best For** | Use when frequent read operations are required. | Use when frequent add/remove operations are needed. |

### **3. What is the difference between Iterator and ListIterator?**

| **Feature** | **Iterator** | **ListIterator** |
| --- | --- | --- |
| **Traversal** | Allows **forward** traversal only. | Allows **forward and backward** traversal. |
| **Modification** | Can remove elements. | Can add, modify, and remove elements. |
| **Applicable To** | Works with all collection types. | Works only with List types (e.g., ArrayList, LinkedList). |
| **Index Access** | Does not provide index access. | Provides index information (nextIndex() and previousIndex()). |

### **4. What is the difference between Iterator and Enumeration?**

| **Feature** | **Iterator** | **Enumeration** |
| --- | --- | --- |
| **Introduced In** | Java 1.2 | Java 1.0 |
| **Traversal** | Traverses in a fail-fast manner. | Traverses in a fail-safe manner. |
| **Modifications** | Can remove elements during traversal. | Does not allow modifications. |
| **Usage** | Preferred for modern collections. | Mostly used for legacy collections like Vector and Hashtable. |

### **5. What is the difference between List and Set?**

| **Feature** | **List** | **Set** |
| --- | --- | --- |
| **Order** | Maintains insertion order. | Does not guarantee order (except LinkedHashSet). |
| **Duplicates** | Allows duplicates. | Does not allow duplicates. |
| **Implementation** | Examples: ArrayList, LinkedList. | Examples: HashSet, TreeSet. |
| **Access** | Index-based access is supported. | No index-based access. |

### **6. What is the difference between HashSet and TreeSet?**

| **Feature** | **HashSet** | **TreeSet** |
| --- | --- | --- |
| **Order** | Does not guarantee order. | Maintains elements in sorted order. |
| **Performance** | Faster for add, remove, and search. | Slower due to sorting overhead. |
| **Null Values** | Allows one null value. | Does not allow null if natural ordering is used. |
| **Implementation** | Backed by a hash table. | Backed by a red-black tree. |

### **7. What is the difference between Array and ArrayList?**

| **Feature** | **Array** | **ArrayList** |
| --- | --- | --- |
| **Size** | Fixed size, defined at declaration. | Dynamic size, grows as needed. |
| **Type** | Can hold primitives and objects. | Holds only objects. |
| **Performance** | Faster as it's a simple data structure. | Slower due to dynamic resizing overhead. |
| **Flexibility** | Cannot be resized. | Can add/remove elements dynamically. |
| **Methods** | No built-in methods for operations. | Provides methods like add(), remove(), contains(). |

**Example:**

**Array:**java  
int[] arr = new int[5];

arr[0] = 1;

**ArrayList:**java  
ArrayList<Integer> list = new ArrayList<>();

* list.add(1);