* Any group of individual objects which are represented as a single unit is known as the collection of the objects.
* A framework is a set of [classes](https://www.geeksforgeeks.org/classes-objects-java/) and [interfaces](https://www.geeksforgeeks.org/interfaces-in-java/) which provide a ready-made architecture.



1. **ArrayList :**

* An ArrayList is a re-sizable array, also called a dynamic array. It grows its size to accommodate new elements and shrinks the size when the elements are removed.
* ArrayList internally uses an array to store the elements. Just like arrays, It allows you to retrieve the elements by their index.
* Java ArrayList allows duplicate and null values.
* Java ArrayList is an ordered collection. It maintains the insertion order of the elements.
* You cannot create an ArrayList of primitive types like int, char etc. You need to use boxed types like Integer, Character, Boolean etc.
* Java ArrayList is not synchronized. If multiple threads try to modify an ArrayList at the same time, then the final outcome will be non-deterministic. You must explicitly synchronize access to an ArrayList if multiple threads are gonna modify it.
* To create an ArrayList, First need to create an object of the ArrayList class. ArrayList contains 3 types of constructors in Java 8.

1. **ArrayList()**: This constructor is to initialize an empty List.
2. **ArrayList(int capacity):** In this constructor, we can pass capacity as a parameter, used to initialize the capacity by the user.
3. **ArrayList(Collection c):** In this constructor, we can pass a Collection c as a parameter, In which an Array list will contain the elements of Collection c.

How ArrayList Works?

* ArrayList is an array of the Object class that is defined as:

**transient Object[] elementData;**

* When an ArrayList is created, its default capacity or size is 10 if not provided by the user. **The size of the ArrayList grows based on load factor and current capacity.**
* The Load Factor is a measure to decide when to increase its capacity. The default value of load factor of an ArrayList is 0.75f
* ArrayList in Java expands its capacity after each threshold which is calculated as the product of *current capacity* and *load factor* of the ArrayList instance.

**Threshold = (Load Factor) \* (Current Capacity)**

* **For example**, if the user creates an ArrayList of size 10,

**Threshold = Load Factor \* Current Capacity**

**= 0.75 \* 10**

**≅ 7**

This means after adding the 7th element to the list, the size will increase as it has reached the threshold value. Internally, a new Array with a new capacity is created and the elements present in the old Array are copied in the new Array.

* the new capacity of the ArrayList is calculated to be **50%** more than its old capacity.

**new\_capacity = old\_capacity + (old\_capacity >> 1)**

* **For example**, if the array size is 10 and it has reached the threshold value, we have to increase its capacity to add new elements. The new capacity will be **10 + (10 >> 1) => 10 + 5 => 15**. Hence, the size is increased from 10 to 15.

**Time Complexities of key ArrayList operations:**

* Random access takes O(1) time
* Adding element takes amortized constant time O(1)
* Inserting/Deleting takes O(n) time
* Searching takes O(n) time for unsorted array and O(log n) for a sorted one

1. **LinkedList:**

* Java LinkedList is a [**doubly linked list**](https://en.wikipedia.org/wiki/Doubly_linked_list) implementation of Java’s List and Deque interfaces. It is part of Java’s collections framework.
* Java LinkedList maintains the insertion order of the elements.
* LinkedList can have duplicate and null values.
* The LinkedList class implements [Queue](https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html) and [Deque](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html) interfaces. Therefore, it can also be used as a Queue, Deque or Stack.
* Java LinkedList is not thread-safe. You must explicitly synchronize concurrent modifications to the LinkedList in a multi-threaded environment.
* The following are the constructors available in this class:  
   1. LinkedList(): This constructor is used to create an empty linked list.

2. LinkedList(Collection C): This constructor is used to create an ordered list that contains all the elements of a specified collection, as returned by the collection’s iterator

**How Does LinkedList Class in Java works internally?**

* Unlike arrays where we have to specify the size of the array before declaration. In Linked List, we don't have to specify the size of the list as a linked list is a dynamic data structure and it automatically changes size when an element is added or removed. Also, the nodes of the linked list are not stored in a contiguous memory location, they are linked to each other with the help of next and previous pointers.

1. **Vector**:

* Vector is like the dynamic array which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit.
* Elements can be of all primitive types like int, float, Object etc.
* It is recommended to use the Vector class in the thread-safe implementation only. If you don't need to use the **thread-safe** implementation, you should use the ArrayList, the ArrayList will perform better in such case.
* The Iterators returned by the Vector class are fail-fast. In case of concurrent modification, it fails and throws the **ConcurrentModificationException**.
* It is similar to the ArrayList, but with two differences-
* **Vector is synchronized**
* Java Vector contains many legacy methods that are not the part of a collections framework

**Vectors Constructors:**

1. **Vector(int initialCapacity, int Increment)** – Constructs a vector with given initialCapacity and its Increment in size.
2. **Vector(int initialCapacity)** – Constructs an empty vector with given initialCapacity. In this case, Increment is zero.
3. **Vector()** – Constructs a default vector of capacity 10.
4. **Vector(Collection c)** – Constructs a vector with a given collection, the order of the elements is same as returned by the collection’s iterator.

**Most Common Errors in Declaration of Vectors:**

* Vector throws an **IllegalArgumentException** if the InitialSize of the vector defined is negative.
* If the specified collection is null, It throws **NullPointerException**

***Note:***

1. If the vector increment is not specified then its capacity will be doubled in every increment cycle.
2. The capacity of a vector cannot be below the size, it may equal to it.

https://www.callicoder.com/java-linkedlist/