PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

INGENIERÍA EN SISTEMAS

P.O.O



CONSULTA

Grupo 1

27/04/2025

**INTRODUCCIÓN**

En el mundo del desarrollo en Java, trabajar con colecciones de datos es una habilidad esencial para crear programas ordenados y funcionales. La plataforma Java proporciona un conjunto de herramientas, conocido como Collections Framework, que facilita la gestión de grupos de objetos de forma flexible y eficiente. Entre las estructuras más utilizadas se encuentran List, LinkedList, Vector y ArrayList, cada una con sus propias ventajas y características específicas.

Este material describe de forma clara y práctica las principales funciones, atributos y constructores de estas clases, detallando también el comportamiento de los métodos más importantes. El objetivo es ofrecer una guía comprensible que permita al lector conocer cuándo y cómo usar cada una de estas estructuras de manera adecuada en diferentes escenarios de programación.

**1. List (Interface)**

List es una interfaz en java.util, que define una colección ordenada que puede contener elementos duplicados. No se puede instanciar directamente; sirve como contrato para clases como ArrayList, LinkedList, y Vector.

**DATOS MIEMBRO:**

Como es una interfaz, no tiene atributos (solo constantes públicas si acaso).

**FUNCIONES MIEMBRO (MÉTODOS ABSTRACTOS MÁS IMPORTANTES):**

* void add(E e) — Agrega un elemento al final.
* void add(int index, E element) — Inserta un elemento en la posición indicada.
* E get(int index) — Obtiene el elemento en una posición.
* E set(int index, E element) — Reemplaza el elemento en una posición.
* E remove(int index) — Elimina el elemento en una posición.
* boolean remove(Object o) — Elimina el primer elemento que coincida.
* boolean contains(Object o) — ¿Contiene un elemento?
* int size() — Número de elementos.
* void clear() — Elimina todos los elementos.
* int indexOf(Object o) — Devuelve el índice de la primera ocurrencia.
* boolean isEmpty() — Verifica si está vacío.

**CONSTRUCTORES:**

No tiene porque es una interfaz. (Solo implementaciones como ArrayList, LinkedList, etc. tienen).

**2. LinkedList (Class)**

LinkedList es una clase concreta que implementa List usando una lista doblemente enlazada.

**DATOS MIEMBRO PRINCIPALES: (PRIVADOS INTERNAMENTE)**

* int size — número de elementos.
* Node<E> first — referencia al primer nodo.
* Node<E> last — referencia al último nodo.

**FUNCIONES MIEMBRO PRINCIPALES:**

* void add(E e) — Agrega un elemento al final.
* void addFirst(E e) — Agrega un elemento al inicio de la lista.
* void addLast(E e) — Agrega un elemento al final (equivalente a add(E e)).
* E get(int index) — Devuelve el elemento en la posición dada.
* E getFirst() — Devuelve el primer elemento.
* E getLast() — Devuelve el último elemento.
* E remove(int index) — Elimina el elemento en la posición indicada.
* E removeFirst() — Elimina el primer elemento.
* E removeLast() — Elimina el último elemento.
* boolean contains(Object o) — Verifica si el objeto existe en la lista.
* int size() — Número de elementos.
* void clear() — Borra todos los elementos.

**CONSTRUCTORES:**

* LinkedList() — Crea una lista vacía.
* LinkedList(Collection<? extends E> c) — Crea una lista inicializada con los elementos de una colección.

**3. Vector (CLASE)**

Vector es una clase antigua (desde JDK 1.0) que implementa List. Es como ArrayList, pero sincronizado (thread-safe).

**DATOS MIEMBRO PRINCIPALES:**

* Object[] elementData — arreglo que contiene los elementos.
* int elementCount — número de elementos.
* int capacityIncrement — cuánto incrementar el tamaño cuando se llena.

**FUNCIONES MIEMBRO PRINCIPALES:**

* void addElement(E obj) — Añade al final.
* void insertElementAt(E obj, int index) — Inserta en índice.
* E elementAt(int index) — Devuelve el elemento en índice.
* void removeElementAt(int index) — Elimina en índice.
* boolean contains(Object o) — Retorna true si el objeto está en el vector.
* int size() — Número de elementos actuales.
* void clear() — Vacía el vector.

**CONSTRUCTORES:**

* Vector() — Capacidad inicial de 10.
* Vector(int initialCapacity) — Capacidad inicial definida.
* Vector(int initialCapacity, int capacityIncrement) — Capacidad inicial y de incremento definidos.
* Vector(Collection<? extends E> c) — A partir de otra colección.

**4. ArrayList (CLASE)**

ArrayList es una clase concreta que implementa List usando un arreglo dinámico.

**DATOS MIEMBRO PRINCIPALES:**

* Object[] elementData — arreglo donde guarda los elementos.
* int size — número de elementos actuales.

**FUNCIONES MIEMBRO PRINCIPALES:**

* boolean add(E e) — Agrega un elemento al final.
* void add(int index, E element) — Inserta un elemento en una posición específica.
* E get(int index) — Devuelve el elemento en el índice dado.
* E set(int index, E element) — Reemplaza el elemento en la posición dada.
* E remove(int index) — Elimina el elemento en el índice especificado.
* boolean remove(Object o) — Elimina la primera aparición de un objeto.
* boolean contains(Object o) — Comprueba si el objeto está en la lista.
* int size() — Retorna el número de elementos.
* void clear() — Elimina todos los elementos de la lista.
* int indexOf(Object o) — Retorna la posición de la primera aparición del objeto.
* boolean isEmpty() — Retorna true si la lista no tiene elementos.

**CONSTRUCTORES:**

* ArrayList() — Crea un array con capacidad 10 por defecto.
* ArrayList(int initialCapacity) — Crea un array con una capacidad inicial definida.
* ArrayList(Collection<? extends E> c) — Crea un array con los elementos de una colección.

**CONCLUSIÓN**

Dominar el uso de las diferentes colecciones que ofrece Java es una ventaja importante al momento de construir aplicaciones bien organizadas y eficientes. A lo largo de este recorrido, pudimos ver que List, LinkedList, Vector y ArrayList no solo se diferencian en su estructura interna, sino también en la manera en que impactan el rendimiento y la lógica de un programa.

Más allá de conocer sus métodos o características técnicas, el verdadero valor está en saber cuándo conviene usar cada una según las necesidades específicas del proyecto. Elegir correctamente puede ayudar a que nuestras aplicaciones sean más rápidas, más seguras o simplemente más fáciles de mantener.

En definitiva, no se trata solo de aprender listas de funciones, sino de entender cómo aprovecharlas de forma inteligente en los distintos escenarios de programación que se nos presenten. La experiencia, combinada con este tipo de conocimiento, es lo que finalmente lleva a un mejor diseño de software.