Material descargable

egg

# Java Collections Framework

## ¿Qué es una Colección?

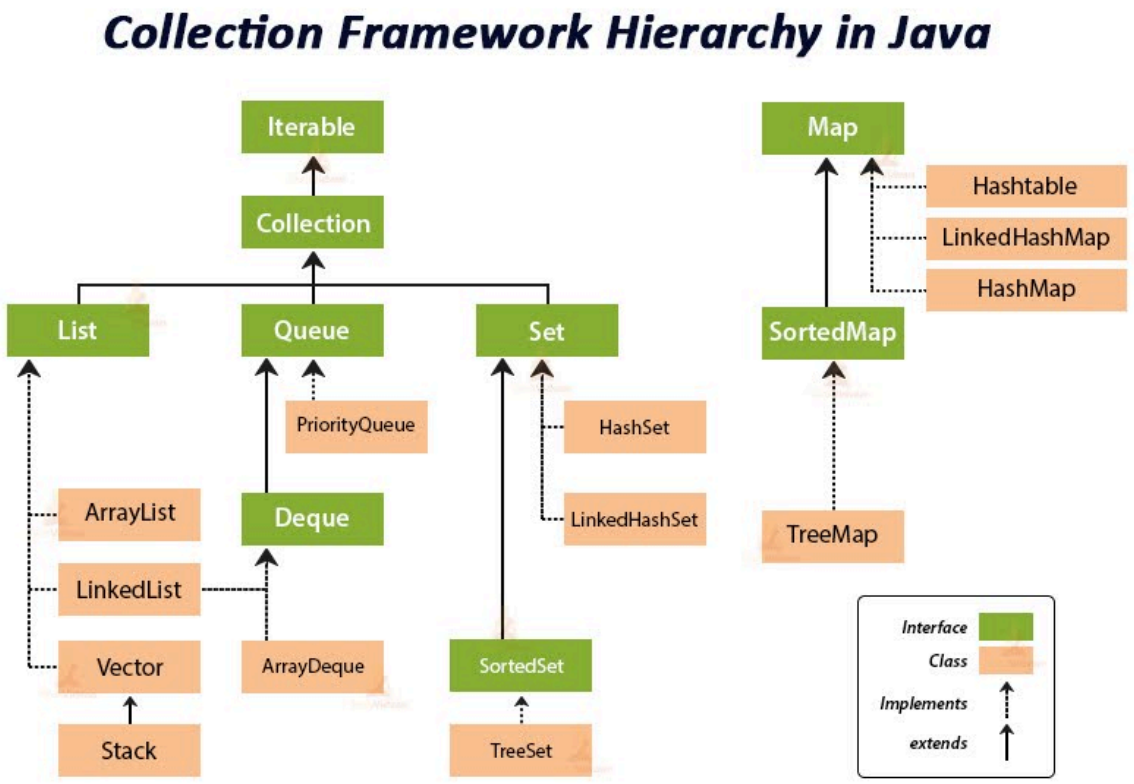
Una colección en programación es un objeto que agrupa múltiples elementos en una sola unidad para luego poder manipularlos de manera conjunta. En Java, las colecciones son una parte fundamental del lenguaje y se utilizan para almacenar, organizar y manipular conjuntos de datos de manera eficiente.

## Beneficios de Usar Colecciones en Java en Lugar de Arrays

- **Tamaño Dinámico:** A diferencia de los arrays, que tienen un tamaño fijo, las colecciones en Java pueden crecer o disminuir dinámicamente según sea necesario. Esto proporciona flexibilidad al trabajar con conjuntos de datos de tamaño variable.
- **Facilidad de Uso:** Las colecciones en Java proporcionan una interfaz más intuitiva y fácil de usar en comparación con los arrays. Ofrecen una amplia gama de métodos y operaciones predefinidas para agregar, eliminar, buscar y ordenar elementos, lo que simplifica el manejo de datos en el código.
- **Polimorfismo:** Las colecciones en Java son polimórficas, lo que significa que podemos trabajar con diferentes tipos de colecciones a través de una interfaz común. Esto facilita la adaptación del código a diferentes escenarios sin necesidad de cambiar su estructura interna.
- **Seguridad de Tipo:** Las colecciones en Java proporcionan seguridad de tipo, lo que significa que solo pueden contener elementos del tipo especificado en su declaración. Esto evita errores de tipo en tiempo de ejecución y hace que el código sea más robusto y seguro.
- **Algoritmos Optimizados:** Java proporciona implementaciones optimizadas de algoritmos para trabajar con colecciones, como la búsqueda, la ordenación y la eliminación de elementos duplicados. Estos algoritmos están diseñados para ofrecer un rendimiento óptimo y una eficiencia en el uso de recursos.

## Explorando la Jerarquía de Interfaces del Java Collections Framework

El Framework de Colecciones de Java proporciona una serie de interfaces, clases y enums predefinidos que representan diferentes tipos de colecciones y algoritmos para manipularlas.




Algunas de las interfaces principales en el framework incluyen:

- **Collection:** Define operaciones comunes para trabajar con colecciones en general, como agregar, eliminar y recorrer elementos.
- **List:** Representa una colección ordenada secuencialmente que permite duplicados y proporciona acceso basado en índices a sus elementos.
- **Set:** Representa una colección que no permite duplicados y no mantiene un orden específico de los elementos.
- **Map:** Asocia claves únicas con valores en una estructura de llave-valor.

# Métodos utilizados con mayor frecuencia (Collection y Collections)

En esta sección se explican algunos métodos importantes asociados a las colecciones:

**Métodos de la Interfaz Collection**

| Método | Tipo de Dato Retornado | Descripción |
|---|---|---|
| `add(E e)` | `boolean` | Agrega un elemento a la colección. |
| `remove(Object o)` | `boolean` | Elimina un elemento de la colección. |
| `clear()` | `void` | Remueve todos los elementos de la colección. |
| `size()` | `int` | Devuelve el número de elementos en la colección. |
| `contains(Object o)` | `boolean` | Verifica si la colección contiene un elemento específico. |
| `isEmpty()` | `boolean` | Devuelve `true` si la colección no contiene elementos. |
| `toArray()` | `Object[]` | Convierte la colección en un array de objetos. |

**Métodos de la Clase Collections**

| Método | Descripción |
|---|---|
| `reverse(List<T> lista)` | Invierte el orden de los elementos de una lista. |
| `reverseOrder()` | Retorna un `Comparator` que invierte el orden de los elementos. |
| `fill(List<T> lista, Objeto objeto)` | Reemplaza todos los elementos de la lista con un elemento específico. |

## Iteración sobre Colecciones con Iterators

Los Iterators son objetos que permiten recorrer y acceder a los elementos de una colección de manera secuencial.
Algunos métodos comunes en Iterators incluyen:

- `hasNext()`: Verifica si hay más elementos en la colección.
- `next()`: Obtiene el siguiente elemento en la iteración.

Los Iterators son especialmente útiles si necesitas agregar o eliminar elementos durante la iteración, ya que otros métodos, como los bucles `fori` o `foreach`, no permiten modificar la estructura de la colección mientras se está iterando. Para modificaciones más avanzadas, se utiliza `ListIterator`.

# Profundizando en la Interfaz List

La interfaz `List` en Java representa una colección ordenada secuencialmente que permite duplicados y proporciona un acceso basado en índices a sus elementos. Al igual que los arrays, su índice comienza en 0.

**Métodos Propios de List**

**Métodos Propios de List**

| Método | Descripción | Tipo de Retorno |
|---|---|---|
| `add(int index, E element)` | Inserta el elemento especificado en la posición indicada en esta lista. | `void` |
| `get(int index)` | Devuelve el elemento en la posición especificada de esta lista. | `E` |
| `remove(int index)` | Elimina el elemento en la posición especificada de esta lista. | `E` |
| `sort(List<T> lista)` | Ordena los elementos de una lista en orden ascendente. | `void` |
| `shuffle(List<T> lista)` | Mezcla los elementos de la lista en un orden aleatorio. | `void` |

| | | |
|---|---|---|
| `subList(int fromIndex, int toIndex)` | Devuelve una vista de la porción de esta lista entre los índices indicados. | `List<E>` |

---

### Implementación Popular: ArrayList

`ArrayList` es una de las implementaciones más utilizadas del Framework de Colecciones. Sus características principales incluyen:

- **Crecimiento Dinámico:** El tamaño del array interno crece automáticamente cuando es necesario.
- **Acceso Eficiente:** Proporciona un acceso rápido a los elementos mediante índices.
- **Permite Duplicados:** Sigue la especificación de la interfaz `List`, que permite duplicados.

💡Siempre es recomendable utilizar la **documentación oficial de Oracle** para obtener más detalles y ejemplos prácticos