Material descargable

egg

Java Collections Framework

¿Qué es una Colección?

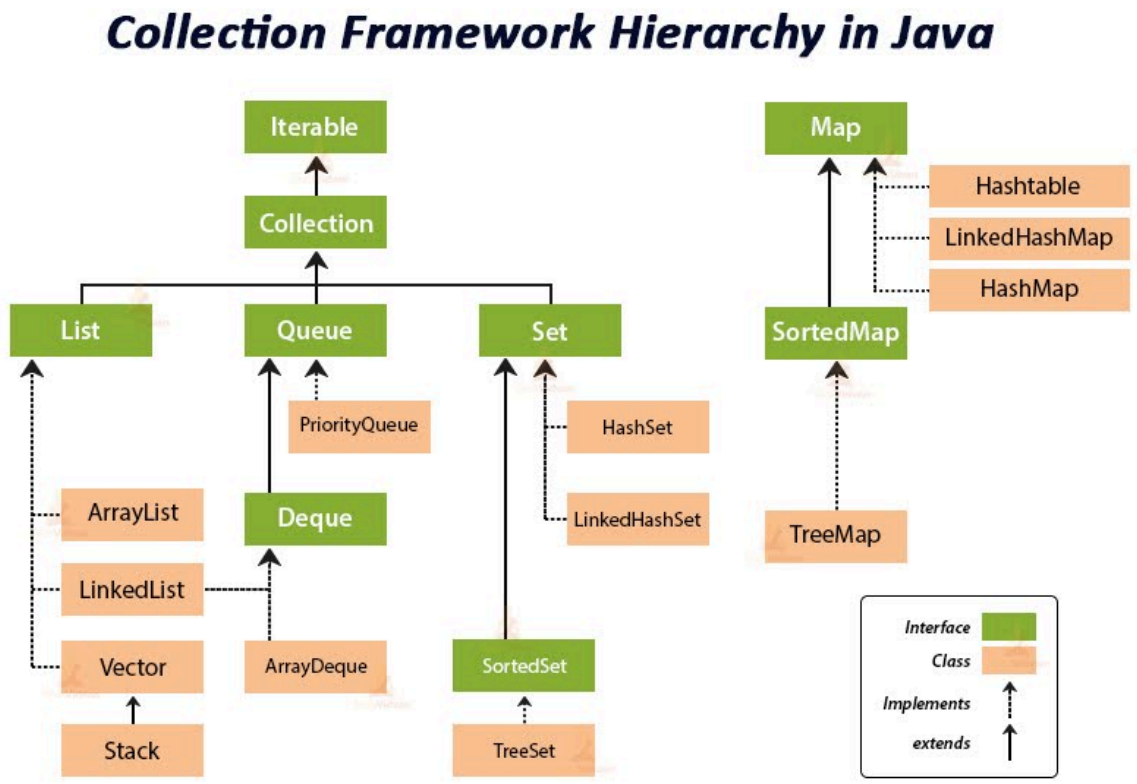
Una colección en programación es un objeto que agrupa múltiples elementos en una sola unidad para luego poder manipularlos de manera conjunta. En Java, las colecciones son una parte fundamental del lenguaje y se utilizan para almacenar, organizar y manipular conjuntos de datos de manera eficiente.

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 de CLASE COLLECTIONS

En este anexo, puedes ver algunos de los distintos métodos explicados brevemente:

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()`		Este método se usa para remover todos los elementos de una lista / conjunto.
`size()`	`int`	Devuelve el número de elementos en la colección.
`contains(Object o)`	`boolean`	Verifica si la colección contiene un elemento específico.
`reverse(List<T> lista)`		Este método invierte el orden de los elementos de una lista.
`reverseOrder()`		Este método retorna un comparator que invierte el orden de los elementos de una colección.
`isEmpty()`	`boolean`	Devuelve `true` si la colección no contiene elementos.
`fill(List<T> lista, Objeto objeto)`		Este método reemplaza todos los elementos de la lista con un elemento específico
`addAll(Collection<? extends E> c)`	`boolean`	Agrega todos los elementos de otra colección a esta colección.

`removeAll(Collection<?> c)`	`boolean`	Elimina todos los elementos de esta colección que también están contenidos en la colección especificada.
`retainAll(Collection<?> c)`	`boolean`	Retiene solo los elementos de esta colección que están contenidos en la colección especificada.
`toArray()`	`Object[]`	Convierte la colección en un array de objetos.
`toArray(T[] a)`	`T[]`	Convierte la colección en un array del tipo especificado.

💡Los métodos de agregar (add) o eliminar (remove) en las colecciones **devuelven un booleano para proporcionar retroalimentación sobre el éxito o el fracaso de la operación.** Este valor booleano indica si la colección ha sido modificada como resultado de la operación realizada. Por ejemplo, si el método add devuelve true, significa que el elemento fue agregado exitosamente a la colección. De manera similar, si el método remove devuelve true, indica que se eliminó exitosamente un elemento de la colección. Esta característica proporciona una forma conveniente para que el programador verifique si la operación se realizó según lo esperado y tome medidas en consecuencia.

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() para verificar si hay más elementos y next() para obtener el siguiente elemento en la iteración.

Un Iterator se utiliza en lugar de los bucles fori o foreach cuando se necesite agregar o eliminar elementos durante la iteración. Esto se debe a que los bucles fori y foreach no permiten modificar la estructura de la colección mientras se está iterando.

Profundizando en 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

Método	Descripción	Tipo de Retorno
`void add(int index, E element)`	Inserta el elemento especificado en la posición indicada en esta lista.	`void`
`boolean addAll(int index, Collection<? extends E> c)`	Inserta todos los elementos de la colección especificada en esta lista en la posición especificada.	`boolean`
`E get(int index)`	Devuelve el elemento en la posición especificada de esta lista.	`E`
`int indexOf(Object o)`	Devuelve el índice de la primera ocurrencia del elemento especificado en esta lista.	`int`
`int lastIndexOf(Object o)`	Devuelve el índice de la última ocurrencia del elemento especificado en esta lista.	`int`
`ListIterator<E> listIterator()`	Devuelve un iterador de lista sobre los elementos de esta lista.	`ListIterator<E>`
`ListIterator<E> listIterator(int index)`	Devuelve un iterador de lista sobre los elementos de esta lista, comenzando en la posición especificada en la lista.	`ListIterator<E>`
`E remove(int index)`	Elimina el elemento en la posición especificada de esta lista.	`E`

`E set(int index, E element)`	Reemplaza el elemento en la posición especificada de esta lista con el elemento especificado.	`E`
`sort(List<T> lista)`	Este método ordena los elementos de una lista de manera ascendente.	`void`
`shuffle(List<T> lista)`	Este método modifica la posición de los elementos de una lista de manera aleatoria	
`List<E> subList(int fromIndex, int toIndex)`	Devuelve una vista de la porción de esta lista entre los índices `fromIndex` (inclusive) y `toIndex` (exclusivo).	

La colección utilizada con mayor frecuencia de esta clase es ArrayList, algunos de sus beneficios más destacados son:

- **Capacidad inicial personalizable:** ArrayList permite especificar una capacidad inicial al crear la instancia, lo que resulta útil cuando se conoce de antemano el tamaño aproximado de la lista. Esta característica puede mejorar el rendimiento y reducir la necesidad de reasignación de memoria conforme la lista crece.

- **Operaciones específicas de ArrayList:** Esta implementación ofrece operaciones exclusivas, como la adición o eliminación de elementos en una posición determinada (por ejemplo, add(index, element) o remove(index)). Si necesitas realizar este tipo de operaciones, una variable de tipo ArrayList resulta más adecuada.

- **Acceso eficiente por índice:** ArrayList almacena los elementos en un arreglo interno, lo que proporciona un acceso eficiente por índice. Este enfoque puede ser beneficioso si planeas acceder frecuentemente a los elementos de la lista mediante sus índices.

- **Permitir elementos duplicados:** Si tu aplicación requiere permitir elementos duplicados en la lista, entonces ArrayList es una opción más apropiada que List. Es importante destacar esta distinción, ya que List es una interfaz que generalmente se utiliza con implementaciones que no permiten duplicados, como Set.

💡Siempre es recomendable utilizar la **documentación oficial** para obtener más detalles sobre el material proporcionado, su uso y su implementación. Puedes acceder a la información sobre Collections Framework desde **aquí**.