Colecciones en Java

La **Colección en Java** es un marco que proporciona una arquitectura para almacenar y manipular el grupo de objetos.

Las colecciones de Java pueden lograr todas las operaciones que realiza en un dato, como búsqueda, clasificación, inserción, manipulación y eliminación.

Colección Java significa una sola unidad de objetos. El marco de Java Collection proporciona muchas interfaces (Set, List, Queue, Deque) y clases (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

¿Qué es la colección en Java?

Una colección representa una sola unidad de objetos, es decir, un grupo.

¿Qué es un framework en Java?

- Proporciona arquitectura prefabricada.
- Representa un conjunto de clases e interfaces.
- Es opcional-

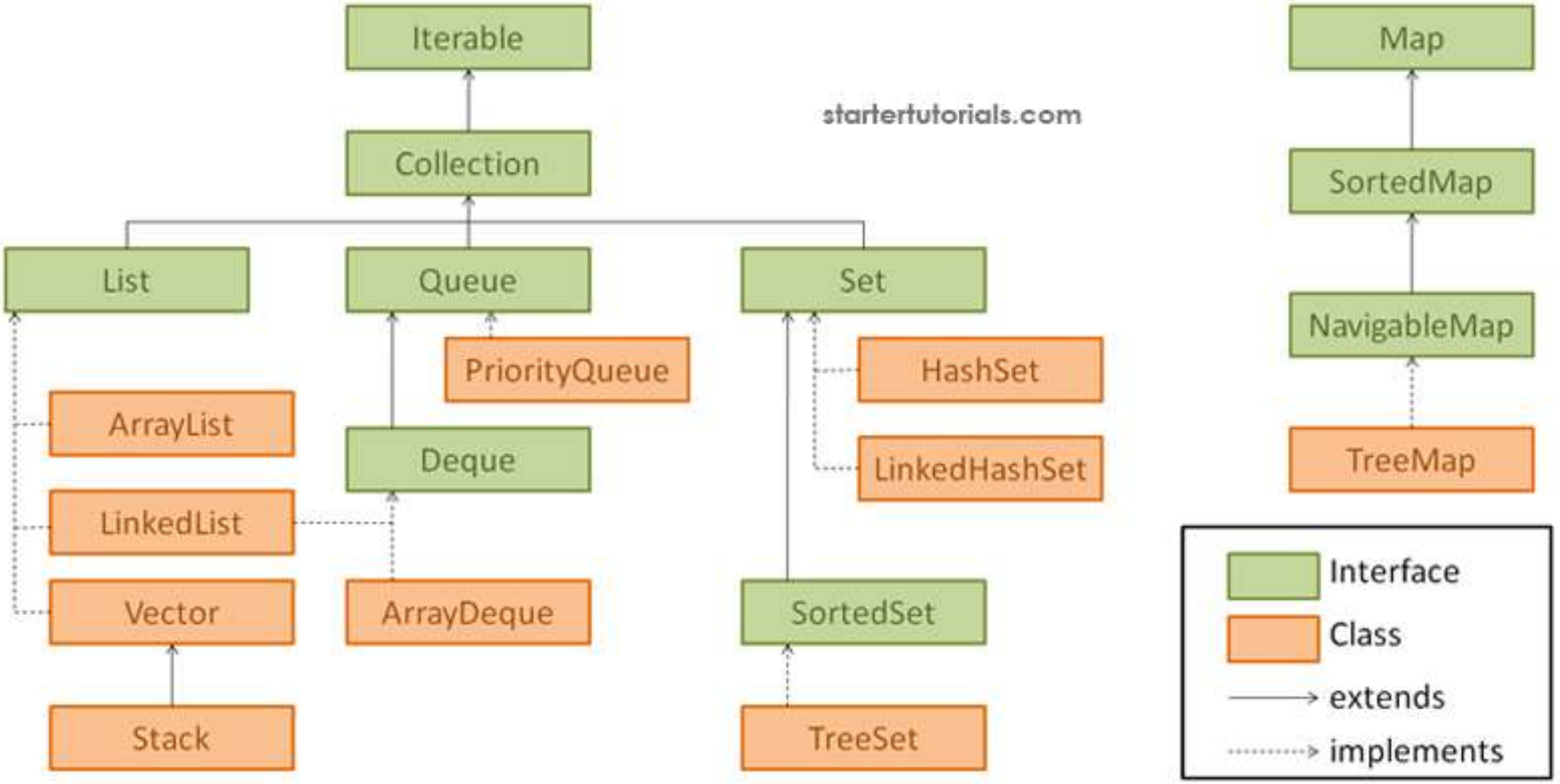
¿Qué es el marco colección?

El marco Colección representa una arquitectura unificada para almacenar y manipular un grupo de objetos. Tiene:

1. Interfaces y sus implementaciones, es decir, clases
2. Algoritmos

Jerarquía del marco Colección

Veamos la jerarquía del marco de Colección. El paquete **java.util** contiene todas las clases e interfaces para el marco de la Colección.



Métodos de interfaz Colección.

No.	Método	Descripción
1	add booleano público (E e)	Se utiliza para insertar un elemento en esta colección.
2	addAll público booleano (Colección <? extiende E> c)	Se utiliza para insertar los elementos de colección especificados en la colección que invoca.
3	public boolean remove (elemento Object)	Se utiliza para eliminar un elemento de la colección.
4 4	public boolean removeAll (Colección <?> c)	Se utiliza para eliminar todos los elementos de la colección especificada de la colección que invoca.
5 5	removeIf booleano predeterminado (filtro Predicate <? super E>)	Se utiliza para eliminar todos los elementos de la colección que satisfacen el predicado especificado.
6 6	public boolean retenciónTodo (Colección <?> c)	Se utiliza para eliminar todos los elementos de la colección de invocación, excepto la colección especificada.
7 7	public int size ()	Devuelve el número total de elementos en la colección.
8	vacío público claro ()	Elimina el número total de elementos de la colección.
9 9	public boolean contiene (Elemento objeto)	Se usa para buscar un elemento.
10	public boolean contiene Todos (Colección <?> c)	Se utiliza para buscar la colección especificada en la colección.
11	Iterator público Iterator ()	Devuelve un Iterator.

12	Objeto público [] toArray ()	Convierte la colección en una matriz.
13	público <T> T [] toArray (T [] a)	Convierte la colección en una matriz. Aquí, el tipo de tiempo de ejecución de la matriz devuelta es el de la matriz especificada.
14	public boolean isEmpty ()	Comprueba si la colección está vacía.
15	Stream predeterminado <E> parallelStream ()	Devuelve un flujo posiblemente paralelo con la colección como fuente.
dieciséis	flujo predeterminado <E> flujo ()	Devuelve una secuencia secuencial con la colección como fuente.
17	Spliterator predeterminado <E> spliterator ()	Genera un Spliterator sobre los elementos especificados en la colección.
18 años	public boolean equals (elemento Object)	Coincide con dos colecciones.
19	public int hashCode ()	Devuelve el número de código hash de la colección.

Interfaz Iterator

La interfaz Iterator proporciona la facilidad de iterar los elementos solo en dirección hacia adelante.

Métodos de interfaz Iterator

No.	Método	Descripción
1	public boolean hasNext ()	Devuelve verdadero si el Iterator tiene más elementos; de lo contrario, devuelve falso.
2	Objeto público next ()	Devuelve el elemento y mueve el puntero del cursor al siguiente elemento.
3	public void remove ()	Elimina los últimos elementos devueltos por el Iterator. Es menos usado.

Interfaz Iterable

La interfaz Iterable es la interfaz raíz para todas las clases de colección. La interfaz de Collection amplía la interfaz Iterable y, por lo tanto, todas las subclases de la interfaz de Collection también implementan la interfaz Iterable.

Contiene solo un método abstracto. es decir,

1. Iterator <T> iterator ()

Devuelve el Iterator sobre los elementos de tipo T.

Interfaz de colección

La interfaz de la Colección es la interfaz que implementan todas las clases en el marco de la colección. Declara los métodos que tendrá cada colección. En otras palabras, podemos decir que la interfaz de la Colección construye la base de la cual depende el marco de la colección.

Algunos de los métodos de la interfaz Collection son Boolean add (Object obj), Boolean addAll (Collection c), void clear (), etc., implementados por todas las subclases de la interfaz Collection.

Interfaz List

La interfaz List es la interfaz secundaria de la interfaz de Colección. Inhibe una estructura de datos de tipo lista en la que podemos almacenar la colección ordenada de objetos. Puede tener valores duplicados.

La interfaz List se implementa mediante las clases ArrayList, LinkedList, Vector y Stack.

Para crear una instancia de la interfaz List, debemos usar:

1. List <data-type> list1 = **new** ArrayList ();
2. List <data-type> list2 = **new** LinkedList ();
3. List <data-type> list3 = **new** Vector ();
4. List <data-type> list4 = **new** Stack ();

Existen varios métodos en la interfaz de la Lista que se pueden usar para insertar, eliminar y acceder a los elementos de la lista.

Las clases que implementan la interfaz Lista se dan a continuación.

ArrayList

La clase ArrayList implementa la interfaz List. Utiliza una matriz dinámica para almacenar el elemento duplicado de diferentes tipos de datos. La clase ArrayList mantiene el orden de inserción y no está sincronizada. Se puede acceder aleatoriamente a los elementos almacenados en la clase ArrayList. Considere el siguiente ejemplo.

Lista enlazada

LinkedList implementa la interfaz de la Colección. Utiliza una lista doblemente vinculada internamente para almacenar los elementos. Puede almacenar los elementos duplicados. Mantiene el orden de inserción y no está sincronizado. En LinkedList, la manipulación es rápida porque no se requiere desplazamiento.

Vector

Vector utiliza una matriz dinámica para almacenar los elementos de datos. Es similar a ArrayList. Sin embargo, está sincronizado y contiene muchos métodos que no forman parte del marco de la Colección.

Pila

La pila es la subclase de Vector. Implementa la estructura de datos de último en entrar, primero en salir, es decir, Pila. La pila contiene todos los métodos de la clase Vector y también proporciona sus métodos como boolean push (), boolean peek (), boolean push (object o), que define sus propiedades.

Interfaz Queue

La interfaz de cola mantiene el orden de primero en entrar, primero en salir. Se puede definir como una lista ordenada que se utiliza para contener los elementos que están a punto de procesarse. Hay varias clases como PriorityQueue, Deque y ArrayDeque que implementa la interfaz Queue.

La interfaz de cola se puede instanciar como:

```
1. Queue <String> q1 = new PriorityQueue ();
2. Queue <String> q2 = new ArrayDeque ();
```

Hay varias clases que implementan la interfaz de cola, algunas de ellas se dan a continuación.

Interfaz Deque

La interfaz de Deque extiende la interfaz de la cola. En Deque, podemos eliminar y agregar los elementos de ambos lados. Deque representa una cola de doble extremo que nos permite realizar las operaciones en ambos extremos.

ArrayDeque

La clase ArrayDeque implementa la interfaz Deque. Nos facilita usar el Deque. A diferencia de la cola, podemos agregar o eliminar los elementos de ambos extremos.ArrayDeque es más rápido que ArrayList y Stack y no tiene restricciones de capacidad.

Set Interface

Está presente en el paquete java.util. Extiende la interfaz de la Colección. Representa el conjunto desordenado de elementos que no nos permite almacenar los elementos duplicados. Podemos almacenar como máximo un valor nulo en Set. Set está implementado por HashSet, LinkedHashSet y TreeSet.

El conjunto se puede instanciar como:

```
1. Set <data-type> s1 = new HashSet <data-type> ();
2. Set <data-type> s2 = new LinkedHashSet <data-type> ();
3. Set <data-type> s3 = new TreeSet <data-type> ();
```

HashSet

La clase HashSet implementa Set Interface. Representa la colección que utiliza una tabla hash para el almacenamiento. Hashing se utiliza para almacenar los elementos en el HashSet. Contiene artículos únicos.

LinkedHashSet

La clase LinkedHashSet representa la implementación de LinkedList de Set Interface. Extiende la clase HashSet e implementa la interfaz Set. Al igual que HashSet, también contiene elementos únicos. Mantiene el orden de inserción y permite elementos nulos.

Interfaz SortedSet

SortedSet es la alternativa de la interfaz Set que proporciona un orden total de sus elementos. Los elementos de SortedSet se organizan en orden creciente (ascendente). SortedSet proporciona los métodos adicionales que inhiben el orden natural de los elementos.

SortedSet se puede instanciar como:

```
SortedSet <data-type> set = new TreeSet ();
```

TreeSet

La clase Java TreeSet implementa la interfaz Set que usa un árbol para el almacenamiento. Al igual que HashSet, TreeSet también contiene elementos únicos. Sin embargo, el tiempo de acceso y recuperación de TreeSet es bastante rápido. Los elementos en TreeSet almacenados en orden ascendente.