Tabla de Contenidos

- Java ForEach
- Java Collections y Jerarquía
- El concepto de Iterador
- Java ForEach Arrays y flexibilidad
- Iterables e Implementación
- Otros artículos relacionados

El uso de Java forEach es uno de los más habituales , ya que continuamente estamos recorriendo colecciones de objetos y la forma de recorrerlos es mucho más comoda usando una instrucción forEach que un bucle clásico. Vamos a ver algunos ejemplos y curiosidades de los que dispone. Empecemos por el bucle clásico:

```
List<String> lista= new ArrayList<String>();
lista.add("hola");
lista.add("que");
lista.add("tal");
lista.add("estas");
for (int i=0;i<lista.size();i++) {
        System.out.println(lista.get(i));
}
```

En este caso tenemos una lista y la vamos a recorrer a través del método size que define el tope.

Podemos ver el resultado por la consola:



Java ForEach

El uso de Java ForEach nos permite recorrer la lista de elementos de una forma mas compacta y el código se reduce.

```
for(String cadena :lista) {
        System.out.println(cadena);
}
```

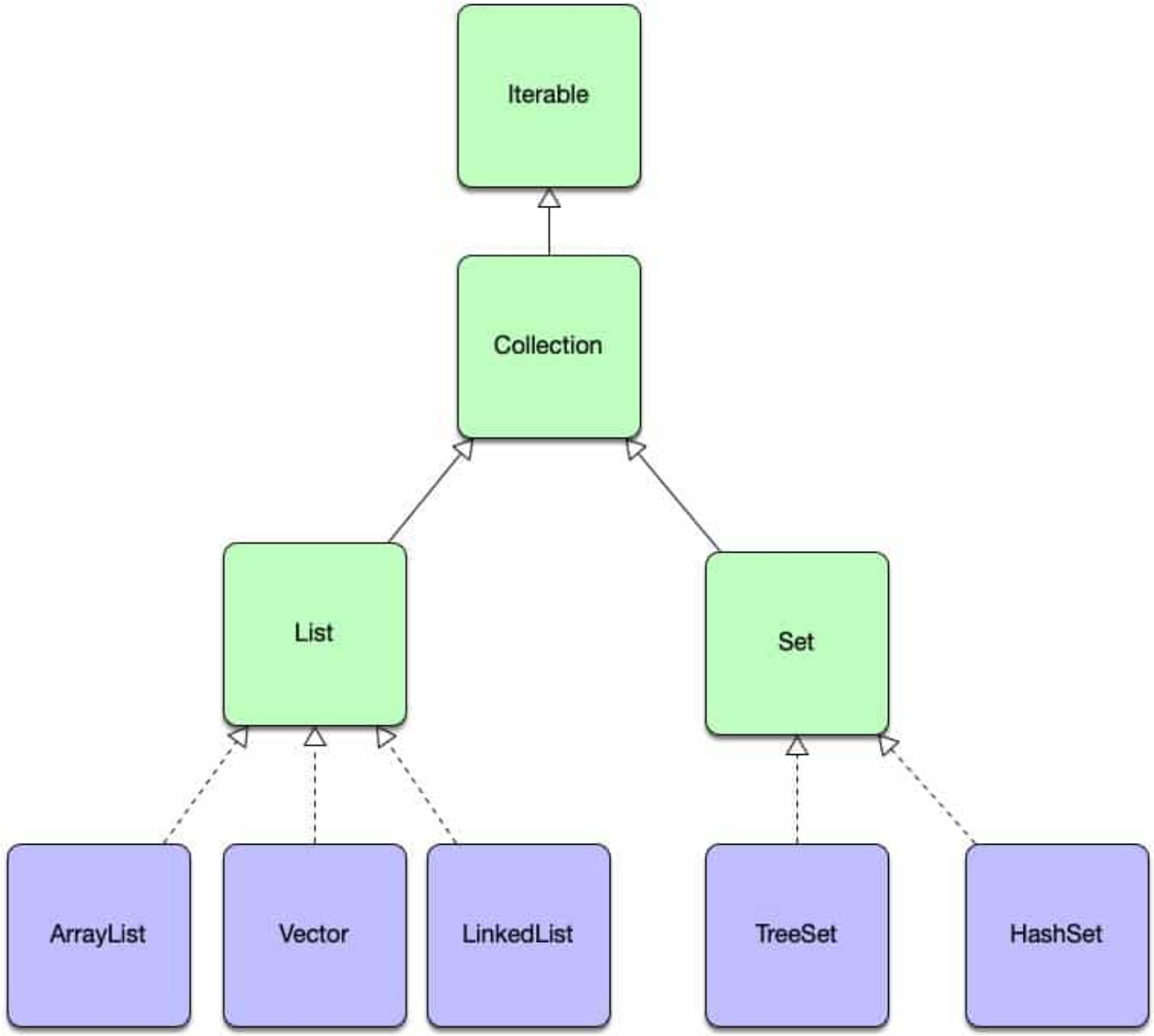
El código es más compacto y el resultado es el mismo



Eso sí la ventaja que tenemos es que no necesitamos saber cual es la propiedad que define el limite de la lista para recorrerla . Es decir no tenemos porque acceder al método size . ¿Como consigue Java que esta operación sea tan natural? . Esto se debe a que el método forEach es una sintaxis sugar sobre el uso de iteradores a nivel del framework de Colecciones .

Java Collections y Jerarquía

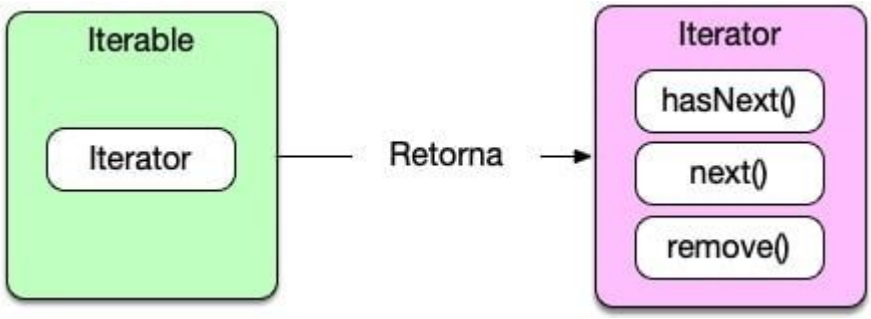
El framework de colecciones define una Jerarquía de clases bastante standard y que ha sido copiada por otros lenguajes de programación debido a su exito.



¿Como se consigue recorrer la colección de elementos de forma tan sencilla? . Esto se debe a que una gran parte de las Colecciones de Java heredan del interface Collection y este a su vez del interface Iterable. El interface Iterable es el que permite recorrer una lista cualquiera sin acceder por posición .

El concepto de Iterador

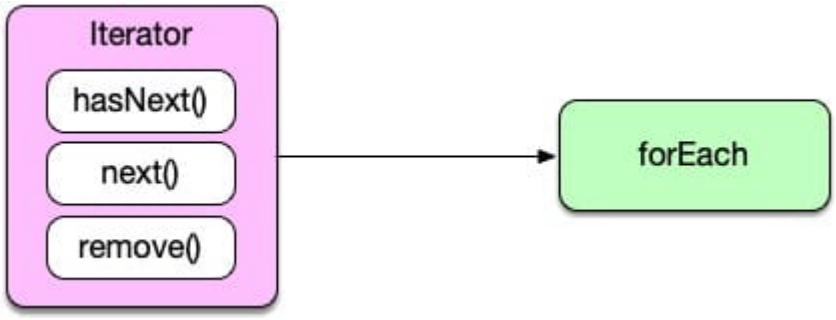
¿Cómo lo hace? . Esto se debe a que el interface Iterable define un método que devuelve un Iterator para una colección determinada.



¿Que es un iterador? . Es un patrón de diseño que nos permite recorrer cualquier grupo de elementos sin necesidad de acceder por posición . Vamos a ver como se usa de forma natural:

```
Iterator<String> it = lista.iterator();
while (it.hasNext()) {
        System.out.println(it.next());
}
```

En este caso hemos pedido a la lista que nos devuelva el iterador y que con el iterador si hay más elementos los recorra y los imprima. Es esta funcionalidad la que usa forEach para poder recorrer cualquier colección , con la ventaja de que al ser una mejora sobre la sintaxis clásica pues la simplifica.



Java ForEach Arrays y flexibilidad

El uso de ForEach a ser sintaxis sugar sobre el lenguaje soporta soporta algunas curiosidades una de ellas es que es capaz de recorrer un Array

```
package com.arquitecturajava;

public class Principal2 {

	public static void main(String[] args) {
		String[] lista= new String[]
{"hola","que","tal","estas"};
		for(String cadena:lista) {
			System.out.println(cadena);
		}
	}
}
```

Un Array en principio no parece ser una clase de Java ya que no se define como una clase normal. Sin embargo si mirásemos la especificación del lenguaje esta nos diría que los Arrays son clases que implementan tanto el método Cloneable como Serializable. Hoy en día soporta también el iterable y podemos recorrer el Array con el bucle for.

Iterables e Implementación

Existen multitud de clases que implementan el interface iterable solo hay que ver la documentación del interface a nivel de jdk para ver que incluso clases como SQLException lo implementan de tal forma que puedan iterar sobre una lista de Excepciones que se produzcan a nivel de base de datos.

Otros artículos relacionados

- Java List to Map y el uso de Collectors
- Java Collections Remove con Java 8
- Novedades de Java 8 Collections y Listas