# **Iterador de Java**

Imagina que vas a la biblioteca y quieres encontrar un libro en particular. ¿Cómo lo haces? Si es no ficción, está el Sistema Decimal Dewey. Si es ficción, se ordenan alfabéticamente por su apellido. Lo que sea que estés buscando, en algún momento, tienes que revisar los libros uno por uno hasta que encuentres lo que estás buscando. Cada vez que tiene una colección de cosas en Java, ya sea que esté almacenada en un Array , un ArrayList , una Queue o una List , ¿cómo encuentra un valor particular? En algún momento, tienes que pasar por cada elemento uno a la vez. Esto es para lo que sirve un iterador de Java.

**¿Qué es un iterador en Java?**

Es una forma de ver cada elemento de una colección. Y por colección, nos referimos a cualquier cosa en la clase Colección . Esto incluye:

* ArrayList
* HashSet
* LinkedHashSet
* LinkedList
* PriorityQueue
* Vector
* y muchos otros…

También incluye las diversas interfaces para esas clases, como Set , List , Queue , Dequeue y Sorted Set , por nombrar algunas. Iterator<E> es el método de interfaz pública de la clase Interface Iterator<E> . Se creó en Java 1.2 y reemplazó a la Enumeración como una forma de examinar elementos secuenciales en una colección.

**Por qué no debería usar For Loops como un iterador en Java**

Una de las primeras formas en que a todos se les enseña a iterar a través de una colección en Java es mediante un bucle. Se parece ha esto:

**class** Main {

**public** **static** **void** main(String[] args) {

**int** exampleArray[] = **new** **int**[10];

//fill array with data

**for**(**int** x = 0; x < exampleArray.length; x++) {

System.out.println("Content of element " + x + "is: " + exampleArray[x]);

}

}

}

La salida sería una lista que dice:

Content of element 0 is: 0

Content of element 1 is: 1

Content of element 2 is: 2

etc.…

Esto tiene sus usos, pero ¿qué sucede si la colección no almacena elementos en un sistema basado en índices? Por ejemplo, los Set normalmente no tienen orden. Por lo tanto, es recomendable salir de la práctica de usar un bucle for como un iterador en Java y practicar el uso de la clase Iterator<E> en su lugar. Aquí hay algunos ejemplos de iteradores de Java.

**Cómo usar iterador en Java**

Aquí hay algunos ejemplos de cómo usar el iterador en Java. Cuando se usa la clase de iterador, hay tres formas de recorrer una colección. Puede usar un bucle while() , un bucle for() y un bucle forEach() . Tenga en cuenta que este ciclo for es diferente del que hablamos antes. Aquí están los tres ejemplos diferentes de iteradores de Java. Primero, configuremos la colección para iterar.

**import** java.util.\*; // imports ArrayList, Collection and Iterator

**class** Main {

**public** **static** **void** main(String[] args) {

Collection<String> example = **new** ArrayList<String>();

example.add("Item 1");

example.add("Item 2");

example.add("Item 3");

example.add("Item 4");

}

}

Esta es una colección simple compuesta por un ArrayList al que cargamos cuatro elementos. Ahora veamos los tres métodos de usar la clase Iterator para atravesar ArrayList .

**Bucle while()**

Iterator<String> iterator = example.iterator();

**while** (iterator.hasNext()) {

System.out.println("Element Value= " + iterator.next());

}

Este ciclo while() usa el método booleano .hasNext() de la clase Iterator para verificar si hay un elemento futuro. Si la condición previa es verdadera, entonces procede. Si devuelve falso, el bucle finaliza. La parte clave aquí es que los métodos .hasNext() y .next() hacen una verificación inicial del primer elemento. Si la colección está vacía y no hay un primer elemento, el método devuelve false para .hasNext() y generará una excepción NoSuchElementException para el método .next() .

**En bucle**

**for** (Iterator<String> iterator = example.iterator(); iterator.hasNext();) {

System.out.println("Element Value= " + iterator.next());

}

Esto parece un bucle for más tradicional. Utiliza el método .hasNext() como la parte de verificación y cambio de condición. La inicialización es la llamada al iterador.

**Para: cada bucle**

**for** (String s : example) {

System.out.println("Element Value= " + s);

}

El ciclo For:Each es un ciclo for, pero si no sabe cómo leerlo, puede resultar un poco confuso. La sintaxis de un bucle For:Each es for (tipo\_datos variableName : collectionName){ body} . Este ciclo for:each tiene un par de inconvenientes. Primero, solo puede atravesar la colección en una dirección. En segundo lugar, debe iterar a través de cada elemento. No puedes saltarte ninguno de ellos. Pero como iterador de lista conveniente en Java, es la mejor opción. En el lado positivo, el ciclo for:each es muy fácil de leer y una vez que lo conoce, es difícil equivocarse. Si se pregunta cuál es la salida de los tres bucles iteradores, todos son iguales:

Element Value= Item 1

Element Value= Item 2

Element Value= Item 3

Element Value= Item 4

**Cómo usar un iterador en Java para mapas**

Los mapas son una forma popular de almacenar datos, pero debido a que no amplían Collection, no puede usar los iteradores anteriores para recorrer directamente un mapa. Entonces, ¿cómo se usa un iterador en Java para pasar por Map s y HashMap s? Hay cuatro buenos métodos de iterador de mapas de Java. Los cubriremos individualmente. Primero, carguemos un mapa con una serie de valores.

**import** java.util.\*; //imports Map and HashMap

**class** Main {

**public** **static** **void** main(String[] args) {

Map<String, String> example = **new** HashMap<String, String>();

example.put("alpha", "one");

example.put("beta", "two");

example.put("gamma", "three");

}

}

**Método de iterador Java Hashmap: forEach (acción)**

example.forEach((k,v) -> System.out.println("Key: "+ k + ", Value: " + v));

Este método utiliza una expresión lambda para iterar. El operador lambda es el método forEach() y el cuerpo imprime los valores. Esto utiliza un operador de parámetros múltiples. Este es el método más rápido y limpio para el iterador de mapas en Java 8.

**Método de iterador Java Hashmap: For:Each() Loop**

**for** (Map.Entry<String, String> iterate : example.entrySet()) {

System.out.println("Key: " + iterate.getKey() + ", Value: " + iterate.getValue());

}

Esto usa la sintaxis For:Each para llamar al método entrySet() para devolver un conjunto que tiene la clave y el valor como sus elementos. Además, cuando se usa el método .Entry() , los objetos solo son verdaderos mientras se produce esta iteración.

**Método de iterador Java Hashmap: Map.Entry<k, v>**

Iterator<Map.Entry<String, String>> iterator = example.entrySet().iterator();

**while**(iterator.hasNext()){

Map.Entry<String, String> element = iterator.next();

System.out.println("Key: " + element.getKey() + ", Value: " + element.getValue());

)

Este método vuelve a convertir el mapa en un conjunto para usar el iterador de colecciones y los métodos. Para estos tres métodos iteradores, el retorno se ve así:

Key: alpha, Value: one

Key: beta, Value: two

Key: gamma, Value: three

Java Hashmap Iterator Method: keySet() and values()

**for** (String key : example.keySet()) {

System.out.println("Key: " + key);

}

**for** (String value : example.values()) {

System.out.println("Value: " + value);

}

Este devuelve los elementos de una manera diferente. Primero devolverá todas las Claves en orden y luego todos los Valores:

Key: alpha

Key: beta

Key: gamma

Value: one

Value: two

Value: three