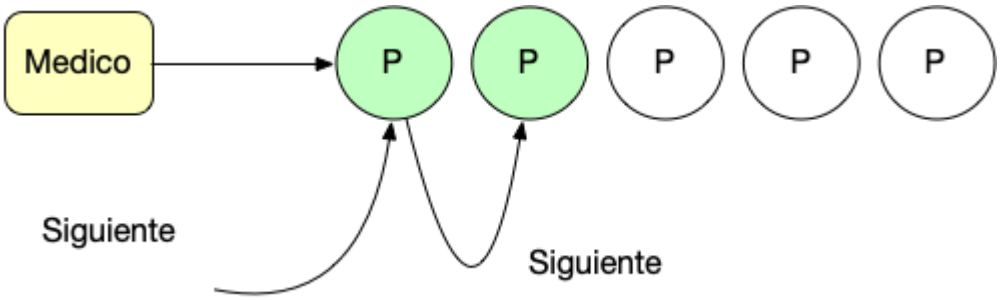
Tabla de Contenidos

- Manejando Java Iterator
- Java 5 y bucle foreach
- Java Iterator y borrado elementos
- Eliminando elementos
- Java Iterator y JSON
  - Conclusiones:

El concepto de Java Iterator es uno de los conceptos que mas cuesta entender a los programadores . ¿Para que sirve un Iterador? . Un Iterador es un objeto que recorre una estructura de datos de forma homogenea. Es decir da lo mismo cual sea la estructura de datos que la forma de recorrer no cambia . Esto nos puede costar entenderlo al principio pero se parece mucho a cuando un médico trata un grupo de pacientes da lo mismo si son un equipo de futbol y están numerados o si son personas que han llegado de forma individual los va tratando uno a uno y pide que pase el siguiente por orden de llegada.



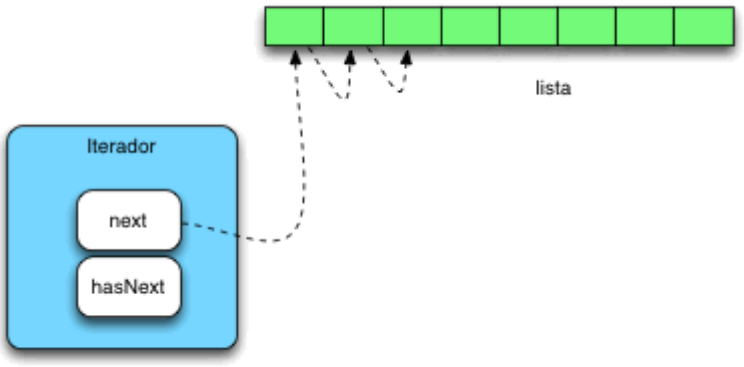
Manejando Java Iterator

Una de las situaciones mas habituales para manejar iteradores es recorrer una lista de elementos. En Java se puede realizar de varias maneras , una es usando un Iterator y la otra o usando un bucle forEach. Vamos a compararlas y ver algunas de las peculiaridades. Vamos a definir una lista de cadenas y la vamos a recorrer a través de un Iterador.

```
ArrayList < String > lista = new ArrayList <String> ();
```

```
lista.add("Pedro");
lista.add("David");
lista.add("Miguel");
lista.add("Antonio");
lista.add("Pedro");
```

Un iterador es un objeto que nos permite recorrer una lista y presentar por pantalla todos sus elementos . Dispone de dos métodos clave para realizar esta operación hasNext() y next().




```
Iterator < String > it = lista.iterator();

while (it.hasNext()) {

  System.out.println(it.next());

}
```

El primero de ellos comprueba si hay un elemento siguiente y el segundo avanza por la colección. Esto nos presentaría la lista de elementos por pantalla :

Pedro
David

Miguel
Antonio
Pedro

Java 5 y bucle foreach

A partir de Java 5 existe otra forma de recorrer una lista que es mucho mas cómoda y compacta , el uso de bucles foreach. Un bucle foreach se parece mucho a un bucle for con la diferencia de que no hace falta una variable i de inicialización :

```
for (String nombre: lista) {

  System.out.println(nombre);
}
```

El resultado es claramente superior y mas legible para todo el mundo . ¿Es momento de dejar de usar iteradores para siempre? . La respuesta del público suele ser que sí . Sin embargo a veces los iteradores aportan cosas interesantes que los bucles foreach no pueden abordar.

Java Iterator y borrado elementos

Vamos a borrar todas las personas que se llaman “Pedro” de la lista . La operación parece tan sencilla como hacer lo siguiente:

```
for (String nombre: lista) {

  if (nombre.equals("Pedro")) {

    lista.remove("Pedro");
  }
```

```
}
```

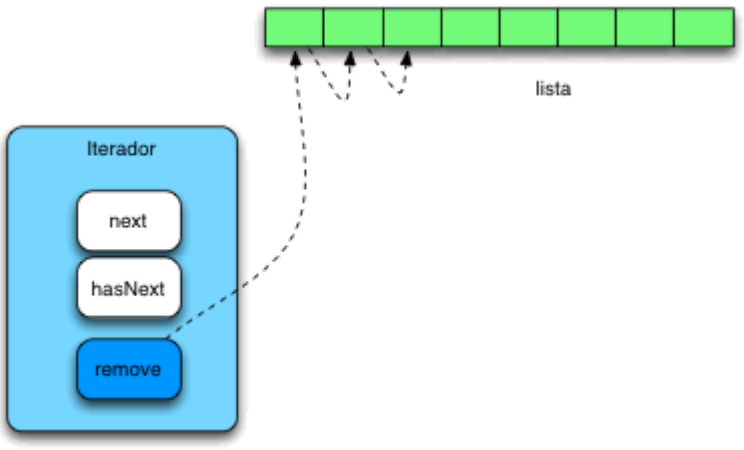
Lamentablemente el código no funciona y lanza una excepción :

```
Exception in thread "main" java.util.ConcurrentModificationException
        at java.util.ArrayList$Itr.checkForComodification(ArrayList.java:819)
        at java.util.ArrayList$Itr.next(ArrayList.java:791)
        at com.arquitecturajava.Principal.main(Principal.java:44)
```

Tenemos un problema ya que la operación que queríamos realizar es sencilla ,pero falla escandalosamente ya que estamos recorriendo y modificando la lista a la vez. Es aquí donde los iteradores pueden venir a rescatarnos.

Eliminando elementos

El interface Iterator dispone de un método adicional que permite eliminar objetos de una lista mientras la recorremos (el método remove) :



Vamos a verlo en ejecución :

```
Iterator < String > it = lista.iterator();

while (it.hasNext()) {
```

```
  String nombre = it.next();
  if (nombre.equals("Pedro")) {

    it.remove();
  }
}
```

El resultado será :

David
Miguel
Antonio

De esta forma podrémos eliminar elementos de una lista de forma segura:) .

Java Iterator y JSON

Los iteradores se pueden usar en muchas áreas de la programación Java por ejemplo algo muy típico es recorrer una estructura JSON. En este caso recorre un conjunto de nodos .

```
package com.arquitecturajava.json5;

import java.io.File;
import java.io.IOException;
import java.util.Iterator;
import java.util.Map.Entry;

import com.fasterxml.jackson.core.exc.StreamWriteException;
import com.fasterxml.jackson.databind.DatabindException;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
```

```
public class Principal2 {

	public static void main(String[] args) {
		try {
			//object mapper es un arbol
			// y ese arbol esta en memoria
			//ocupa espacio pero es muy flexible
			ObjectMapper mapeador= new ObjectMapper();
		   JsonNode nodoRaiz=mapeador.readTree(new
File("datos1.json"));
			 Iterator<Entry<String,JsonNode>>
it=nodoRaiz.fields();
			 while(it.hasNext()) {
				Entry<String,JsonNode>
entrada=it.next();
				System.out.print(entrada.getKey()+
":");
System.out.println(entrada.getValue().asText());
			 }
		} catch (StreamWriteException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (DatabindException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	}
```

```
}
```

En este caso es capaz de recorrer el siguiente fichero JSON.

```
{
        "nombre": "pedro",
        "apellidos": "gomez",
        "edad": "20",
        "estudios": "informatica",
        "experiencia": "true"
}
```

Mostrando los resultados por la consola el nombre de cada uno de los campos:

```
nombre:pedro
apellidos:gomez
edad:20
estudios:informatica
experiencia:true
```

Conclusiones:

El iterador es una forma de recorrer rapidamente cualquier tipo de colección que lo soporte y sus métodos son sencillos de utilizar Java iterator javadoc

- Java Iterable Interface y como implementarlo
- Java forEach y sus opciones
- Java Stream forEach y colecciones
- Uso de Java Generics (I)
- Java Array for y sus trucos