El patrón Iterador es uno de los patrones de la banda de los cuatro (gang of four) .Concretamente se trata de un patrón que se encuentra dentro del ámbito de los patrones estructurales y sirve para recorrer de una forma sencilla una colección de elementos. En muchas ocasiones cuesta entender cual es su funcionamiento real y para que podemos usarlo . Vamos a ver un ejemplo apoyándonos en el API de Java para ver su funcionamiento en una primera toma de contacto.

```
package com.arquitecturajava.ejemplo4;

import java.util.ArrayList;
import java.util.Iterator;

public class Principal {

	public static void main(String[] args) {
		ArrayList<String> lista= new ArrayList<String>();
		lista.add("hola");
		lista.add("que");
		lista.add("tal");
		lista.add("estas");
		Iterator<String> it=lista.iterator();
		// con un iterador
		while(it.hasNext()) {
			System.out.println(it.next());
		}
		/// muy muy inferior y mucho mas claro
		for (int i=0;i<lista.size();i++) {
			System.out.println(lista.get(i));
		}

	}
```

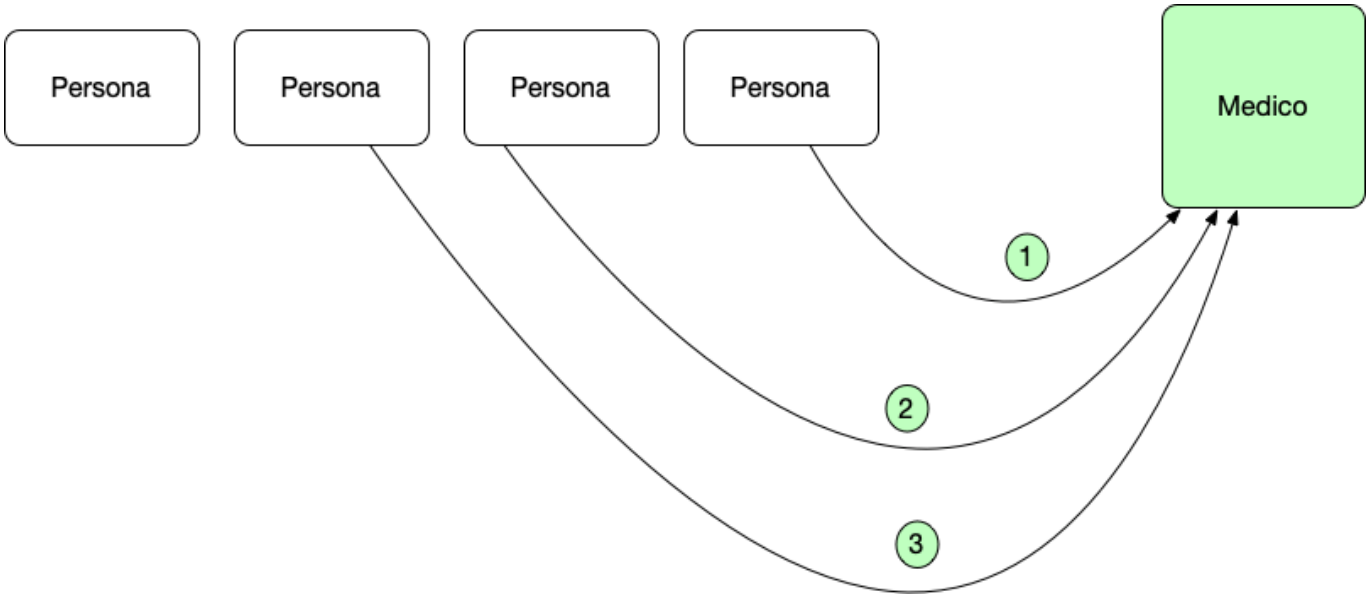
```
}
```

El resultado se ve en la consola:

```
hola
que
tal
estas
hola
que
tal
estas
```

Si nos fijamos acabamos de recorrer una lista de cadenas de dos formas diferentes . La primera usando un iterador (Patrón Iterador) en la cual recorremos la lista solicitando el objeto iterador y luego iterando sobre ella con dos métodos:

- hasNext(): Nos comprueba que lia lista tiene elementos que recorrer
- next(): Nos devuelve el siguiente elemento de la lista a recorrer
  El patrón Iterador se parece mucho al concepto de tener un grupo de gente a los cuales vamos atendiendo uno a uno . Preguntamos primero si hay alguién que atender y en el caso de que haya alguien solicitamos que pase esa persona de una en una .



En este caso el médico ha atendido a tres personas y le queda una por atender. Muchas veces cuando uno empieza a programar en Java no entiende porque la necesidad de manejar iteradores ya que es mucho más sencillo simplemente usar un bucle for clásico.

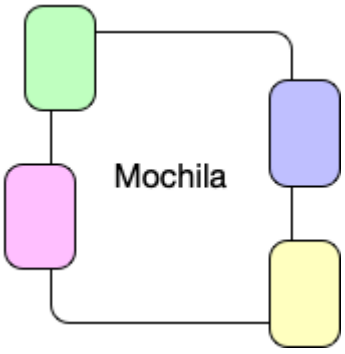
El patrón iterador y el bucle forEach

A partir de Java 5 tenemos la opción de en vez de usar para iterar un Iterador de forma directa usarlo de una forma indirecta a traves de “sintaxis sugar” . Esto simplifica el código de forma clara y lo hace más cómodo de utlizar:

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

El resultado sería el mismo. Por lo tanto ¿Cual es la la ventaja de usar un iterador? . La ventaja es que es un patrón de diseño que nos aisla totalmente de la forma en la que la colección esta construida en el orden que tiene permitiendonos recorrer cualquier cosa de forma sencilla . Vamos a verlo usando el concepto de Mochila . Una mochila tiene varios huecos y nosotros podemos acceder a cada uno de ellos y sacar un objeto del hueco. Sin embargo no se trata de unos huecos ordenados de una forma determinada ni de una lista de

huecos.



Vamos a verlo en código:

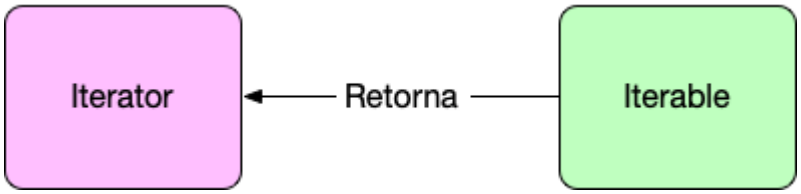
```
package com.arquitecturajava.ejemplo5;

import java.util.Iterator;

public class Mochila implements Iterable<String> {

	// una estructura un poco caotica
	private String zona1;
	private String zona2;
	private String zona3;
	private String zona4;
	private int posicion = 0;
	private int contador = 0;

	public String getZona1() {
		return zona1;
	}

	public void setZona1(String zona1) {
		this.zona1 = zona1;
		contador++;
```

```
}

public String getZona2() {
        return zona2;
}

public void setZona2(String zona2) {
        this.zona2 = zona2;
        contador++;
}

public String getZona3() {
        return zona3;
}

public void setZona3(String zona3) {
        this.zona3 = zona3;
        contador++;
}

public String getZona4() {
        return zona4;
}

public void setZona4(String zona4) {
        this.zona4 = zona4;
        contador++;
}

public Mochila() {
        super();
```

```
	}

	@Override
	public Iterator<String> iterator() {

		return new Iterator<String>() {

			@Override
			public boolean hasNext() {

				if (Mochila.this.posicion >= 4) {
					posicion=0;
					return false;
				} else {
					return true;

				}

			}

			@Override
			public String next() {

				if (Mochila.this.contador != 0) {

					if (Mochila.this.posicion == 0
&& Mochila.this.zona1 != null) {
Mochila.this.posicion++;
						return zona1;
					}
```

```
                                        if (Mochila.this.posicion == 1
&& Mochila.this.zona2 != null) {
Mochila.this.posicion++;
                                                return zona2;
                                        }
                                        if (Mochila.this.posicion == 2
&& Mochila.this.zona3 != null) {
Mochila.this.posicion++;
                                                return zona3;
                                        }
                                        if (Mochila.this.posicion == 3
&& Mochila.this.zona3 != null) {
Mochila.this.posicion=4;
                                                return zona4;
                                        }

                                }

                                throw new IndexOutOfBoundsException();

                        }

                };

        }

}
```

En este caso he construido una clase Mochila que implementa el interface Iterable , este interface devuelve un iterator (patrón iterador) y permite recorrer un objeto con su estructura interna.



Si nos fijamos un poco nos daremos cuenta que la Mochila no contiene una lista de zonas , sino que contiene 4 zonas dispares en donde se pueden ubicar elementos . Lo que he hecho es recorrer cada zona y si no estaba vacía devolver el objeto y pasar a la zona siguiente. De esta forma una estructura que en principio no se puede recorrer pasa a ser facilmente recorrible con un iterador.

```
package com.arquitecturajava.ejemplo5;

import java.util.Iterator;

public class Prioncipal {

	public static void main(String[] args) {
		Mochila m= new Mochila();
		m.setZona1("navaja");
		m.setZona2("bocadillo");
		m.setZona3("agua");
		m.setZona4("telefono");
		Iterator<String> it= m.iterator();
		while(it.hasNext()) {
			System.out.println(it.next());
		}
		for(String zona: m) {
			System.out.println(zona);
		}

	}
}
```

```
navaja
bocadillo
agua
telefono
navaja
bocadillo
agua
telefono
```

Acabamos de recorrer la Mochila sin que esta tuviera una estructura interna clara. Hemos usado el patrón iIterador para abordar esta problemática y el código queda limpio.

Otros artículos relacionados

- Java Collections Framework y su estructura
- Java Iterable to List
- Java forEach y sus opciones
- Java Iterator