MOTIVACIÓN

Los métodos que conllevan el recorrido secuencial de una colección de objetos se utilizan con mucha frecuencia. Las colecciones de objetos se pueden implementar de diversas maneras (ver **Tabla *1***).

Lista basada en array	Lista basada en lista ligada

`public class ListaArray {`
`  private int numElementos;`
`  private String[] lista;`
`  …`
`}`
	`public class Nodo {`
	`  private String dato;`
	`  private Nodo sig;`
	`  …`
	`}`
	
	`public class ListaLigada {`
	`  private Nodo primero;`
	`  …`
	`}`

Tabla 1. Posibles implementaciones de una colección de objetos

Cada implementación concreta de la colección requiere implementar también el método correspondiente para recorrer secuencialmente sus elemente, La**Tabla *2*** muestra cómo se realiza el recorrido secuencial de las colecciones de objetos según la representación interna seleccionada.

Lista basada en array	Lista basada en lista ligada

`…`
`String dato = null;`
`for (int i= 0; i< lista.size();i++) {`
`  dato = lista.get(i);`
`  …`
`}`
`…`
	`…`
	`String dato = null;`
	`Nodo aux = lista.getFirst();`
	`while (aux!=null) {`
	`  dato = aux.getContent();`
	`  …`
	`  aux = aux.getNext();`
	`}`
	`…`

Tabla 2. Fragmentos de código para el recorrido secuencial de la colección de objetos

¿¿¿Tiene sentido que la implementación del recorrido de una colección sea tan distinto cuando se está realizando lo mismo???

OBJETIVO

Proporcionar una forma de acceder a los elementos de una colección de objetos de manera secuencial independiente de la representación interna de dicha colección.

DESCRIPCIÓN

El patrón **Iterador** define una forma homogénea de acceder secuencialmente a los elementos de una colección de objetos independientemente de su representación interna y, por lo tanto, sin exponer la misma.

El patrón **Iterador** se basa en el uso de interfaces que determinan cómo se realiza el recorrido secuencial (**Figura *1***). La interfaz **Iterador** define las operaciones que se pueden utilizar para determinar si todavía quedan elementos sin recorrer en la colección y obtener el siguiente elemento disponible. La interfaz **ColeccionIterable** define el método que permite obtener el **Iterador** que se utilizará para realizar el recorrido secuencial. La colección concreta implementa esta interfaz

basándose en una implementación particular de la interfaz **Iterador**.

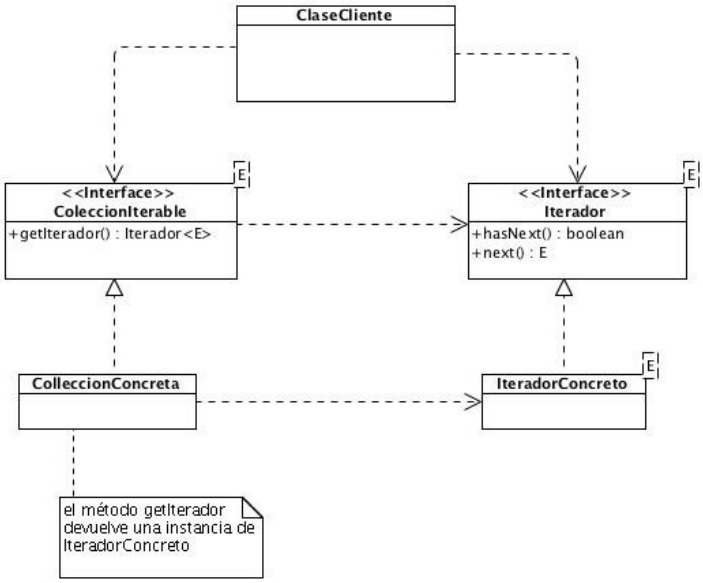

Figura 1. Diagrama UML del patrón **Iterador**.

IMPLEMENTACIÓN

La implementación del patrón **Iterador** se basa en la implementación de las interfaces que definen cómo realizar el recorrido. La implementación concreta del iterador se lleva a cabo, normalmente, en clases privadas internas. A continuación se muestra un fragmento de la implementación de la clase ArrayList en la que se puede observar cómo se aplica el patrón descrito en este documento.

```
public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess,
                    Cloneable, java.io.Serializable {
       ...
       public Iterator<E> iterator() {
              return new Itr();
       }
       private class Itr implements Iterator<E> {
              int cursor;       // index of next element to return
              int lastRet = -1; // index of last element returned; -1 if no such

              public boolean hasNext() {
                     return cursor != size;
              }
              ...
       }
       ...
}
```

UTILIZACIÓN

Este patrón se debe aplicar siempre que se quiera realizar un recorrido secuencial de una colección de objetos (ej., una búsqueda, mostrar el contenido de la lista,…).

VENTAJAS

Al utilizar este patrón, los métodos que realizan el recorrido secuencial de la colección de objetos desconocen la estructura interna de la misma y, por lo tanto, ésta puede ser modificada sin que los métodos que recorren dicha colección se vean afectados.

Ejemplo de uso

La ***Tabla 3*** muestra dos ejemplos de recorridos secuenciales en contenedores de diferente clase utilizando el patrón **Iterador**. Como puede observarse, el proceso es el mismo en ambos casos.

ArrayList	**LinkedList**

`ArrayList<String> lista =`
`  new ArrayList<String>();`
`…`
`String cadena = null;`
`Iterator<String> iter = lista.iterator();`
`while (iter.hasNext()){`
`  cadena = iter.next();`
`  …`
`}`
`…`
	`LinkedList<String> lista =`
	`  new LinkedList<String>();`
	`…`
	`String cadena = null;`
	`Iterator<String> iter = lista.iterator();`
	`while (iter.hasNext()){`
	`  cadena = iter.next();`
	`  …`
	`}`
	`…`

Tabla 3. Ejemplos de recorrido de colecciones de objetos utilizando iteradores