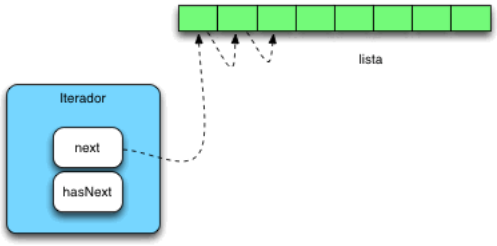
# Patrón de Diseño

## Iterador (iterator)

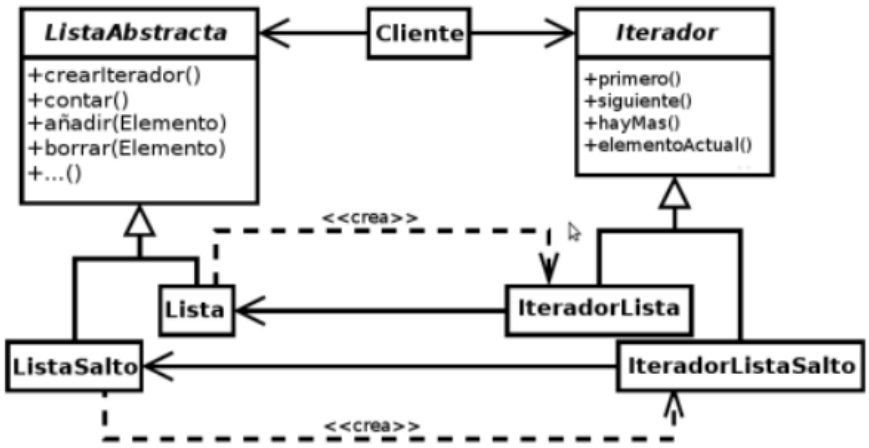
Toda estructura de datos, debería proveer un modo de brindar acceso a sus elementos **sin exponer su estructura interna**. Se debe permitir acceder a objetos individuales de cualquier estructura de datos (listas, vectores, ect) sin tener que conocer el comportamiento de esta estructura.




La idea principal en este patrón es tomar la responsabilidad del acceso y recorrido del contenedor y colocarlo dentro de otra clase (clase ***Iterator)***.

> El patrón **iteador** es un mecanismo de acceso y recorrido a los elementos de una estructura de datos para la utilización de estos sin exponer la su estructura interna de la coleccion.

Para cada estructura de datos se crea un ***Iterator*** específico.




Cada ***Iterator*** específico responde a una misma interfaz común, haciendo esto factible cambiar fácilmente de contenedor o colección.

Un objeto ***Iterador*** es responsable de mantener la pista del elemento actual; es decir sabe cuáles elementos ya han sido recorridos.

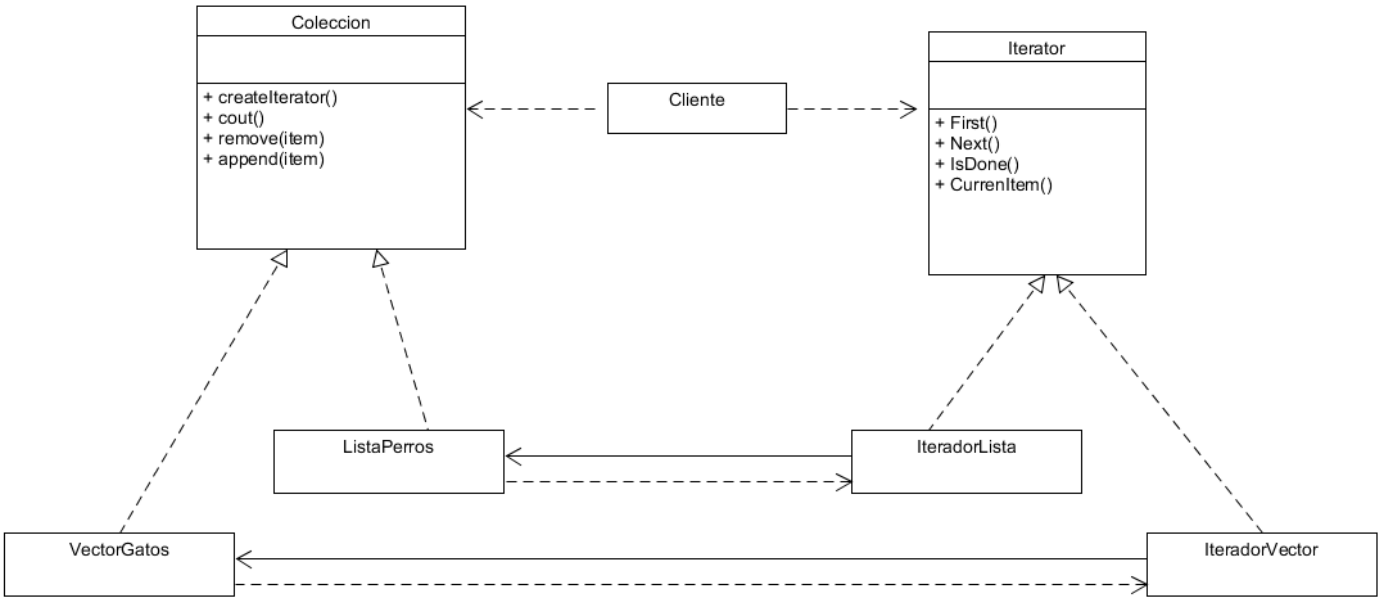
El cliente puede incluso utilizar un ***Iterador*** sin saber sobre qué colección concreta se está ejecutando.

Por tanto el patrón iterator permite el acceso al contenido de una estructura sin exponer su representación interna. Además diferentes iteradores pueden presentar diferentes tipos de recorrido sobre la estructura (recorrido de principio a fin, recorrido con saltos...). Los iteradores no tienen por qué limitarse a recorrer la estructura, sino que podrían incorporar otro tipo de lógica (por ejemplo, filtrado de elementos). Dados diferentes tipos de estructuras, el patrón iterador permite recorrerlas todas utilizando una interfaz común uniforme.

**Metodos básicos del Iterador**

**First()** *// pone el puntero en el primer elemento*
**Next()** *// pone el puntero en el próximo elemento*
**IsDone()** *// retorna true si hay mas elementos y false sino*
**CurrenItem( )** *// retorna el elemento actualmente apuntado*

**Ejemplo caso concreto:**




Cada clase contenedora creará y retornará su iterador específico.

El patron necesita una clase ***abstracta Iterator*** que define una interfaz común de iteración. Entonces podemos definir subclases concretas de iteradores para las implementaciones de diferentes colecciones. Como resultado, el mecanismo de iteración se hace independiente de las colecciones.

**Aplicabilidad**

- Permite acceso total de objetos contenidos sin exponer su representación interna.
- Proporciona una interfaz uniforme para recorrer diferentes estructuras agregadas, independientemente de cual sea esta.
- Evita al cliente estudiar cómo funciona internamente cada colección para poder recorrerla.
- Simplifica el código de las colecciones, los recorridos podrían estar en los iteradores y no en las colecciones
- Gracias a los iteradores es posible tener varios recorridos sobre una misma colección