POLITÉCNICA

"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL

MiW

Patrones de Diseño

17. *Iterator*

Luis Fernández Muñoz

https://www.linkedin.com/in/luisfernandezmunyoz

setillofm@gmail.com

INDICE

1. Problema

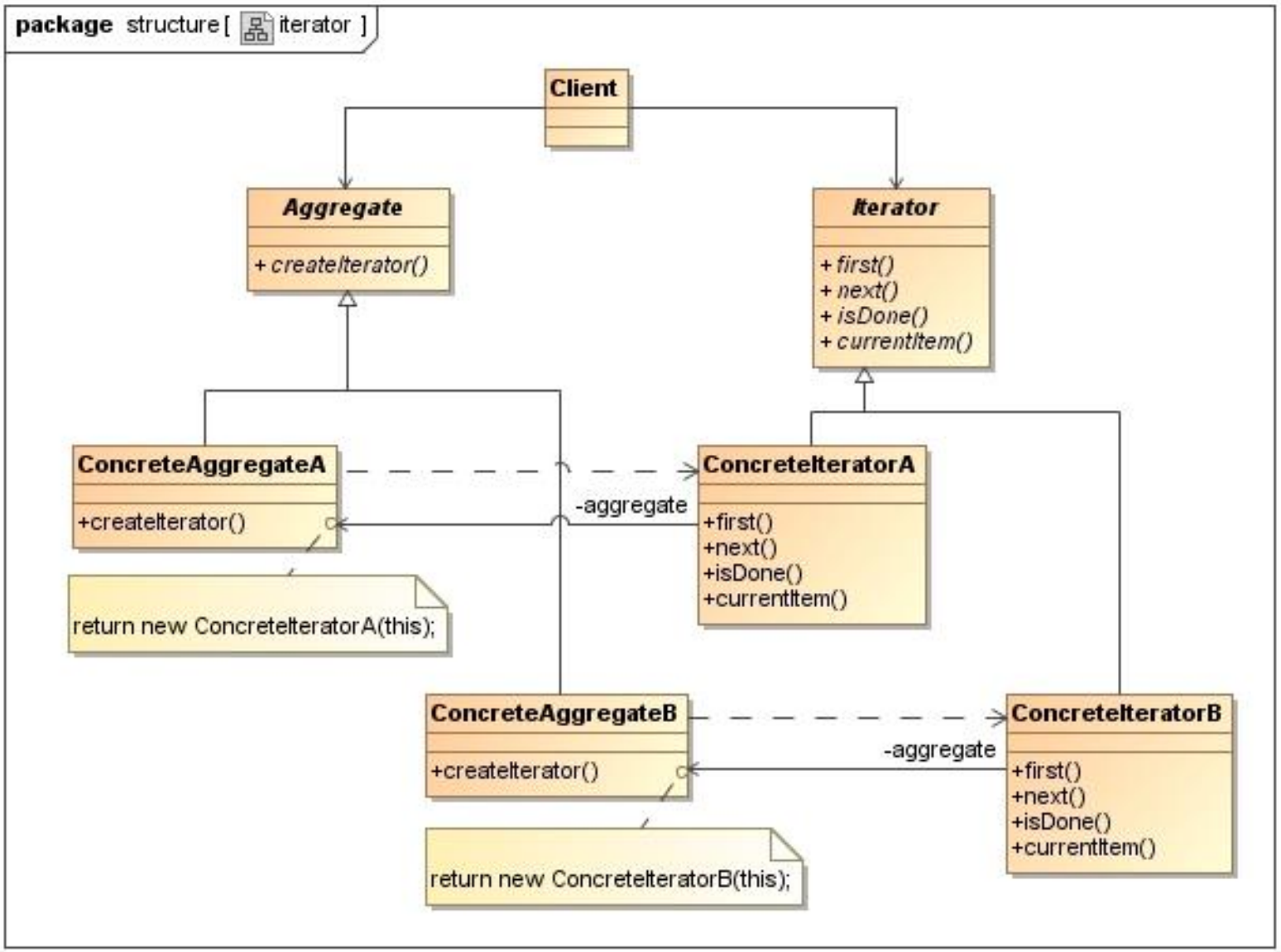
- *En un centro comercial con todos los productos expuestos en los estantes sin dependientes en el local, son los clientes los que tienen que conocer/saber encontrar la localización de los productos. Por ese motivo, cuando cambian la localización de los productos (p.e. obras, campaña de navidad, …) los usuarios no encuentran los productos y quedan afectados.*
- *En un comercio que no sea autoservicio, el dependiente accede "secuencialmente" a los productos de un almacén sin exponer sus estanterías o estructuras de almacenamiento. De esta manera, no se afecta al cliente cuando hay algún cambio de localización de los productos en el almacén, solo afecta al dependiente.*

- *Provee una forma de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación subyacente*

1. Problema

- Un objeto agregado, tal como una lista, debe ofrecer una forma de acceder a sus elementos para un algoritmo del cliente.
  - Es posible que desee recorrer la lista de elementos de diferentes maneras, dependiendo de lo que quiere lograr.
  - Pero es probable que no se quiera inflar la interfaz de la lista con las operaciones para diferentes recorridos, incluso si se pudiera anticipar los que se necesitan.
  - Podría ser necesario tener más de un recorrido en espera en la misma lista.
  - Sería mejor si pudiéramos cambiar la clase agregada sin cambiar el código de cliente

2. Solución


package structure [iterator]
Client
Aggregate
+ createIterator()
Iterator
+ first()
+ next()
+ isDone()
+ currentItem()
ConcreteAggregateA
+createIterator()
return new ConcreteIteratorA(this);
-aggregate
ConcreteIteratorA
+first()
+next()
+isDone()
+currentItem()
ConcreteAggregateB
+createIterator()
return new ConcreteIteratorB(this);
-aggregate
ConcreteIteratorB
+first()
+next()
+isDone()
+currentItem()

2. Solución

- La interfaz minima del iterador son las operaciones *First*, *Next*, *IsDone* y *CurrentItem*. En agregados ordenados podría añadir *Previous*. La operación *SkipTo* es útil para colecciones ordenadas o indexables aplicando algún criterio.
- Cuando el cliente controla la iteración, el iterador se llama un iterador externo y cuando el iterador lo controla, el iterador es un iterador interno. Los clientes que utilizan un iterador externo deben avanzar en el recorrido demandando explícitamente el siguiente elemento del iterador. Por el contrario, el cliente entrega al iterador interno una operación a realizar, y el iterador se aplica la operación a cada elemento del agregado.

2. Solución

- El iterador no es el único lugar en el que el algoritmo de recorrido puede estar definido. Puede ser que el algoritmo de recorrido necesite acceder a las variables privadas del agregado. Si es así, poner el algoritmo de recorrido en el iterador viola la encapsulación del agregado.
  - El agregado puede definir el algoritmo de recorrido y utilizar el iterador para almacenar sólo el estado del recorrido. Llamamos a este tipo de iterador un cursor, ya que se limita a señalar la posición actual en el agregado. Un cliente invocará sobre el agregado la operación *Next* con el cursor como un argumento y la operación *Next* cambiará el estado del cursor

2. Solución

- Un iterador robusto asegura que las inserciones y borrados no interfieren en el recorrido y lo hace sin copiar el agregado.
  - Hay muchas maneras de implementar iteradores robustos. La mayoría lo hacen registrando los iteradores con el agregado. En la inserción o borrado, el agregado o bien ajusta el estado interno de los iteradores que ha producido, o se mantiene información internamente para asegurar el recorrido adecuado.

3. Consecuencias

- Agregados complejos pueden ser atravesados de muchas maneras.
- Varios recorridos pueden estar pendientes el recorrido porque cada Iterador realiza un seguimiento de su propio estado de recorrido.
- Los Iteradores simplifican la interfaz agregada porque la interfaz de recorrido del Iterador obvia la necesidad de una interfaz similar en el agregado.

4. Relaciones

- ***Memento*** para volver al estado anterior en la iteración
- ***Factory Method*** para crear instancias de las subclases de *Iterator*