Iterator

Objective

Provide a way to sequentially access the elements of an aggregated object, without exposing its underlying representation.

Function

According to the Gang of Four book, we can perform tours on composite objects regardless of their implementation.

Structure

As shown in figure 1

This design pattern will be useful to access the elements of an array or collection of objects contained in another object.

The structure that meets this pattern is shown in Figure 1

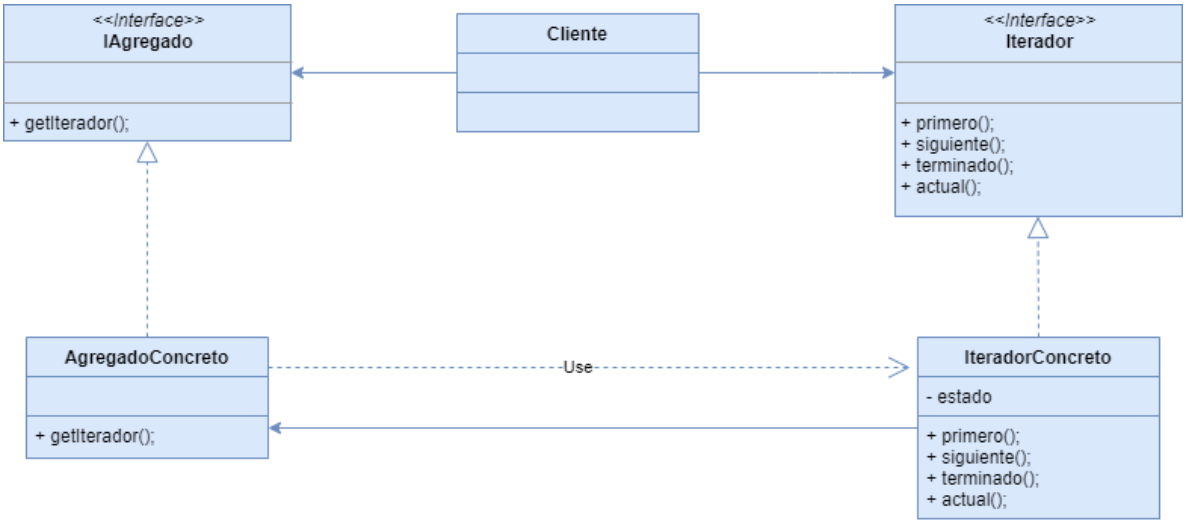

Figure 1: UML Diagram Iterator Pattern

Applications

The use of the Iterator pattern is recommended when:

- It requires handling a collection of objects and there are different ways to navigate through it.
- There are different collections of objects for the same navigation logic.
- Different filters and sorting algorithms can be applied.

Design Patterns Collaborators

- Iterator patterns are generally applied to recursive structures, so the use of the composite pattern is very common.

- The factory method pattern allows you to instantiate the appropriate iterator for a collection.

- The memento pattern is generally used to internally retain the status of each iteration.

Scope of action

Applied at the object level.

Problem

To handle several objects contained in a collection, it must be instantiated with its navigation algorithm; this implies the creation of several objects, ensuring that the client class knows the implementations of them, overloading the memory and causing a high coupling between the objects and the classes that instantiate them.

Solution

The Iterator design pattern implements an interface to navigate through the collection of objects, allowing each one to discriminate the sequence to follow; that is to say that it is enough to instantiate the interface and each object will implement it indicating the current object and the next one, according to some ordering criteria of the elements of the collection.

Diagram or Implementation

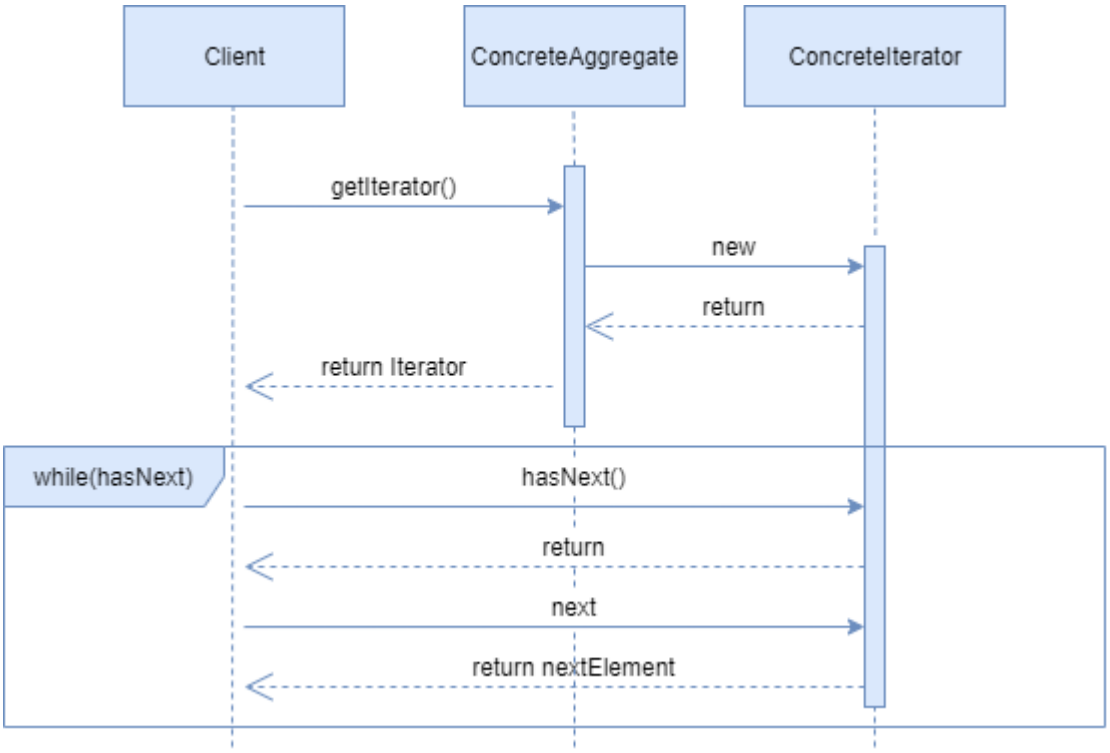

Figure 2: UML Diagram Iterator Pattern

Figure 2 explains the behaviour of the pattern by means of a sequence diagram.

- The client class prompts the ConcreteAggregate component to create an iterator.

- The ConcreteAggregate component creates a new Iterator.

- The client class, to go through the elements, enters a cycle until there are no more elements in the iterator, the hasNext method will tell you when the end has been reached.

- The client class requests the next element from the iterator using the next method.

- If there are more elements we go back to step three, this is repeated until the end of the tour.