Behavioral Design Pattern

Hung Tran

Fpt software

December 3, 2021

Outline

Behavioral Pattern Overview

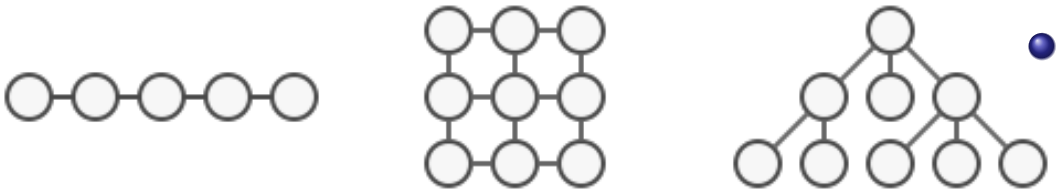
Behavioral design patterns are concerned with algorithms and the assignment of responsibilities between objects.

- **Chain of responsibility**: lets you pass requests along a chain of handlers.
- **Command**: turns a request into a stand-alone object that contains all information about the request.
- **Iterator**: lets you traverse elements of a collection without exposing its underlying representation (list, stack, tree, etc.).
- **Mediator**: lets you reduce chaotic dependencies between objects.
- **Memento**: lets you save and restore the previous state of an object without revealing the details of its implementation.

Behavioral Pattern Overview

- **Observer**: lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing.
- **State**: lets an object alter its behavior when its internal state changes. It appears as if the object changed its class.
- **Strategy**: lets you define a family of algorithms, put each of them into a separate class, and make their objects interchangeable.
- **Template Method**: defines the skeleton of an algorithm in the superclass but lets subclasses override specific steps of the algorithm without changing its structure.
- **Visitor**: lets you separate algorithms from the objects on which they operate.

Problem Statement: Collections

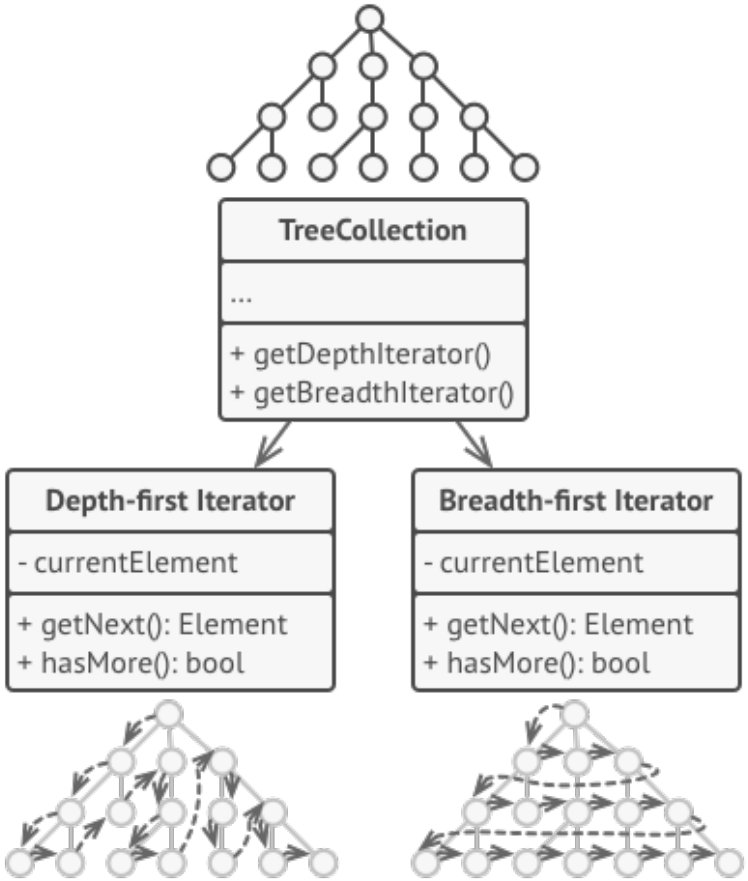


- Collections are based on stacks, trees, graphs and other complex data structures.
- But no matter how a collection is structured, it must provide some way of accessing its elements so that other code can use these elements.

- If you have a collection based on a list, you just loop over all of the elements.
- How do you sequentially traverse elements of a complex data structure, such as a tree?
- Depth-first traversal or breadth-first traversal.

Clients not care how they store their elements?

Solution: Iterator Design Pattern

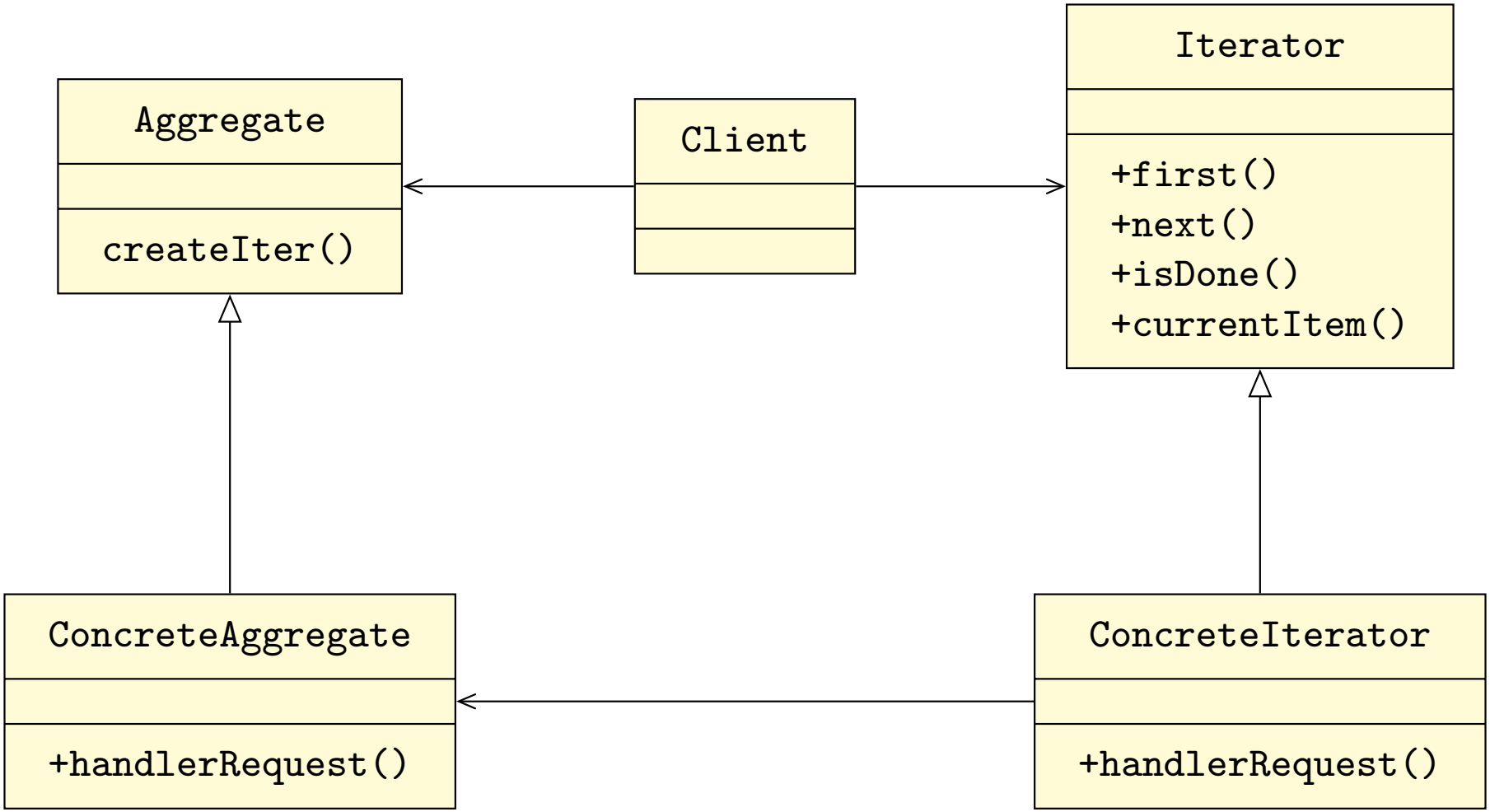


- An iterator object encapsulates all of the traversal details, such as the current position and how many elements are left till the end.
- Several iterators can go through the same collection at the same time, independently of each other.
- All iterators must implement the same interface.

The Intent of Iterator Design Pattern

Command is a behavioral design pattern that turns a request into a stand-alone object that contains all information about the request. This transformation lets you pass requests as a method arguments, delay or queue a request's execution, and support undoable operations.

Structure of Command Pattern



Applicability

- Use the Iterator pattern when your collection has a complex data structure under the hood, but you want to hide its complexity from clients (either for convenience or security reasons).
- Use the pattern to reduce duplication of the traversal code across your app.
- Use the Iterator when you want your code to be able to traverse different data structures or when types of these structures are unknown beforehand.

How to Implement

- Declare the iterator interface. At the very least, it must have a method for fetching the next element from a collection. But for the sake of convenience you can add a couple of other methods, such as fetching the previous element, tracking the current position, and checking the end of the iteration.
- Declare the collection interface and describe a method for fetching iterators.
- Implement concrete iterator classes for the collections that you want to be traversable with iterators.
- Implement the collection interface in your collection classes.
- Go over the client code to replace all of the collection traversal code with the use of iterators.

Pros and Cons

- Single Responsibility Principle. You can clean up the client code and the collections by extracting bulky traversal algorithms into separate classes.
- Open/Closed Principle. You can implement new types of collections and iterators and pass them to existing code without breaking anything.
- You can iterate over the same collection in parallel because each iterator object contains its own iteration state.
- For the same reason, you can
- Applying the pattern can be an overkill if your app only works with simple collections.
- Using an iterator may be less efficient than going through elements of some specialized collections directly.

Relations with Other Patterns

- You can use Iterators to traverse Composite trees.
- You can use Factory Method along with Iterator to let collection subclasses return different types of iterators that are compatible with the collections.
- You can use Memento along with Iterator to capture the current iteration state and roll it back if necessary.
- You can use Visitor along with Iterator to traverse a complex data structure and execute some operation over its elements, even if they all have different classes.

Thank You!