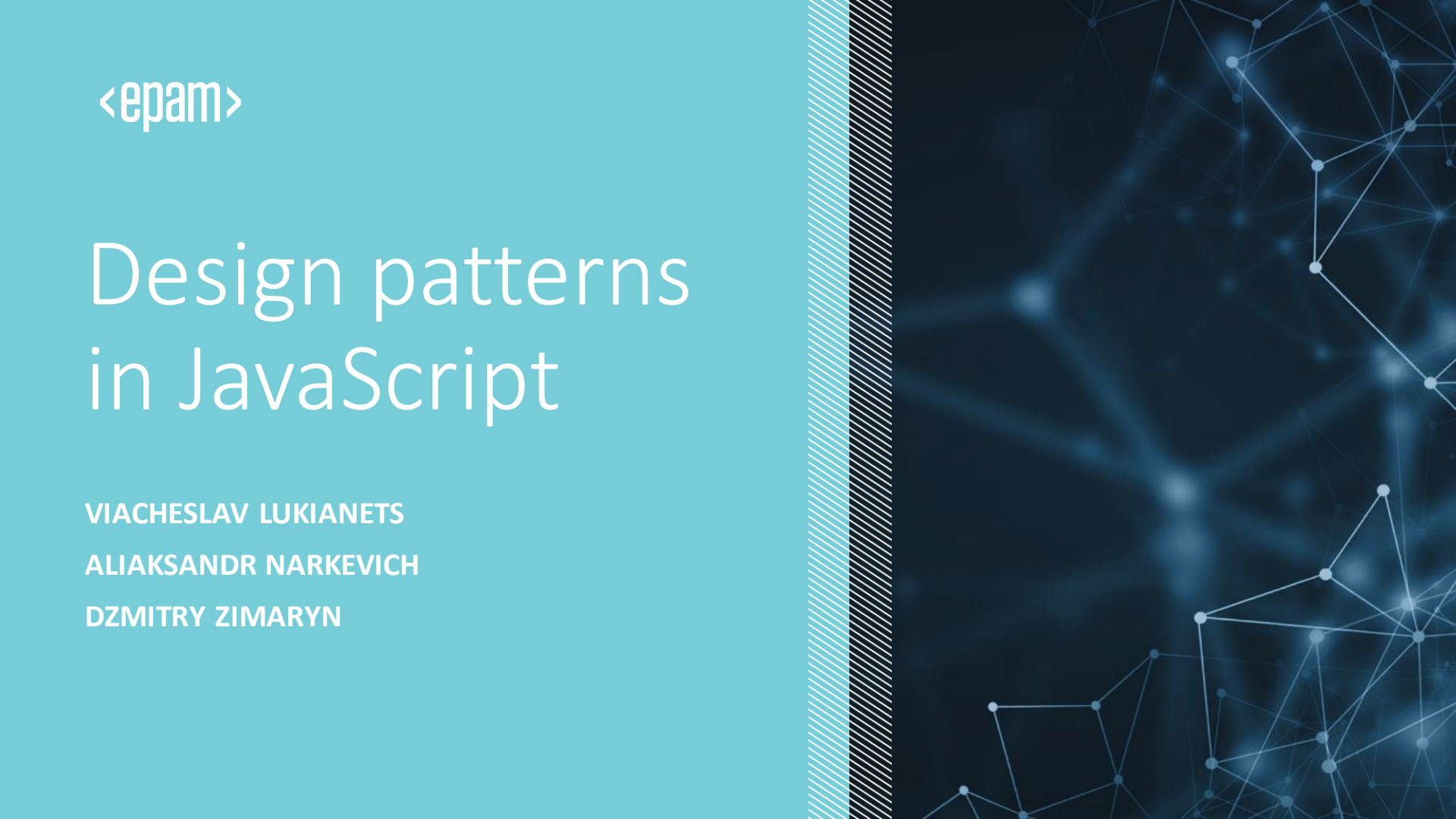


Design patterns in JavaScript

VIACHESLAV LUKIANETS

ALIAKSANDR NARKEVICH

DZMITRY ZIMARYN

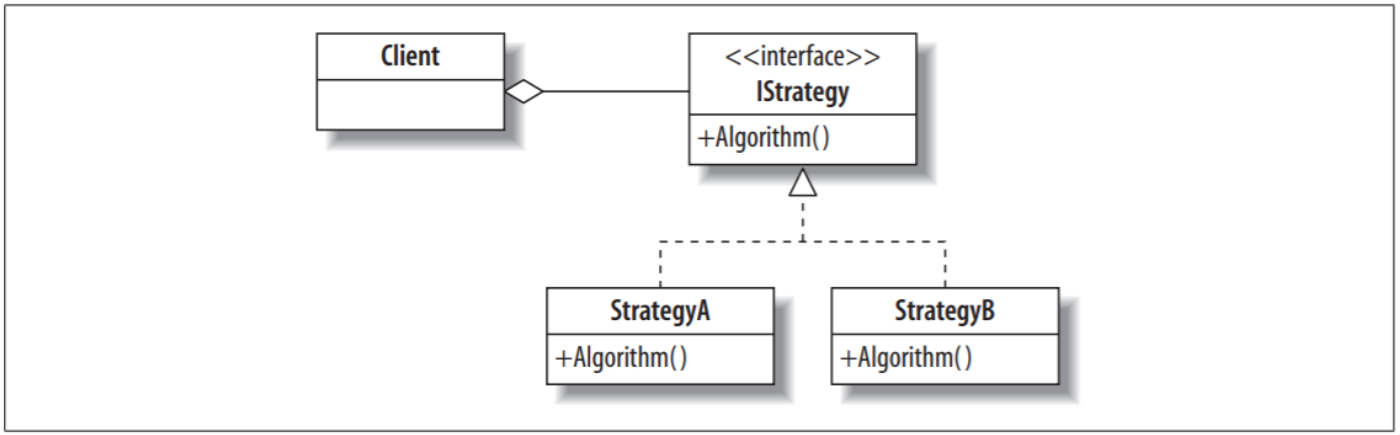
BEHAVIORAL PATTERNS

OVERVIEW

- **Strategy -** involves removing an algorithm from its host class and putting it in a separate class
- **State -** dynamic version of the Strategy pattern
- **Template -** enables algorithms to defer certain steps to subclasses
- **Chain of responsibility -** works with a list of Handler objects that have limitations on the nature of the requests they can deal with
- **Command -** creates distance between the client that requests an operation and the object that can perform it
- **Iterator -** provides a way of accessing elements of a collection sequentially, without knowing how the collection is structured
- **Mediator -** enable objects to communicate without knowing each other's identities
- **Observer -** defines a relationship between objects so that when one changes its state, all the others are notified accordingly
- **Visitor -** defines and performs new operations on all the elements of an existing structure, without altering its classes
- **Interpreter -** supports the interpretation of instructions written in a language or notation defined for a specific purpose
- **Memento -** used to capture an object's internal state and save it externally so that it can be restored later

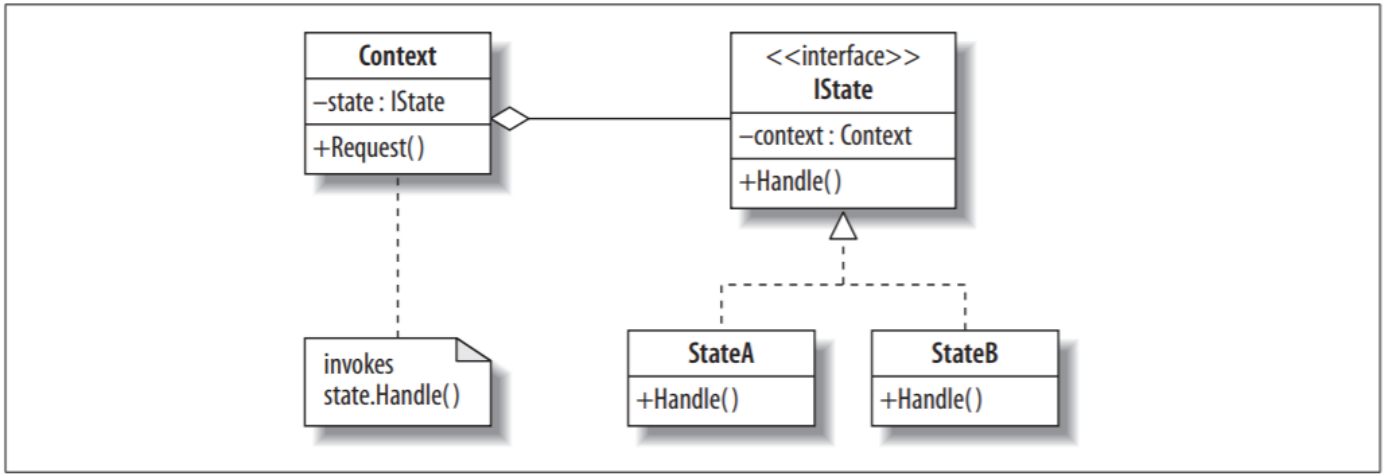
STRATEGY

The Strategy pattern involves removing an algorithm from its host class and putting it in a separate class. There may be different algorithms (strategies) that are applicable for a given problem. If the algorithms are all kept in the host, messy code with lots of conditional statements will result. The Strategy pattern enables a client to choose which algorithm to use from a family of algorithms and gives it a simple way to access it. The algorithms can also be expressed independently of the data they are using.



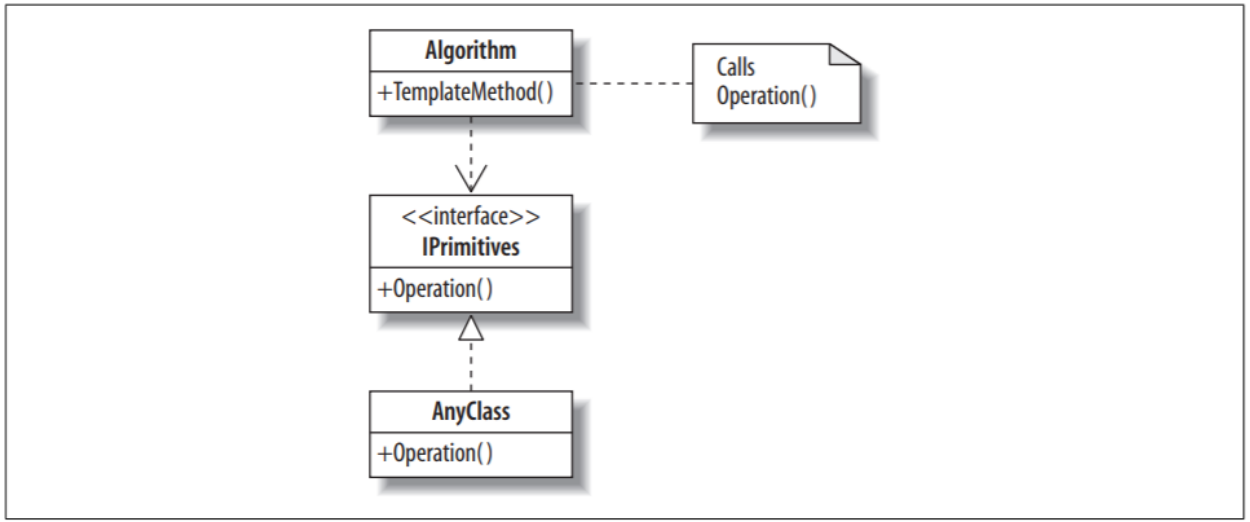
STATE

The State pattern can be seen as a dynamic version of the Strategy pattern. When the state inside an object changes, it can change its behavior by switching to a set of different operations. This is achieved by an object variable changing its subclass, within a hierarchy



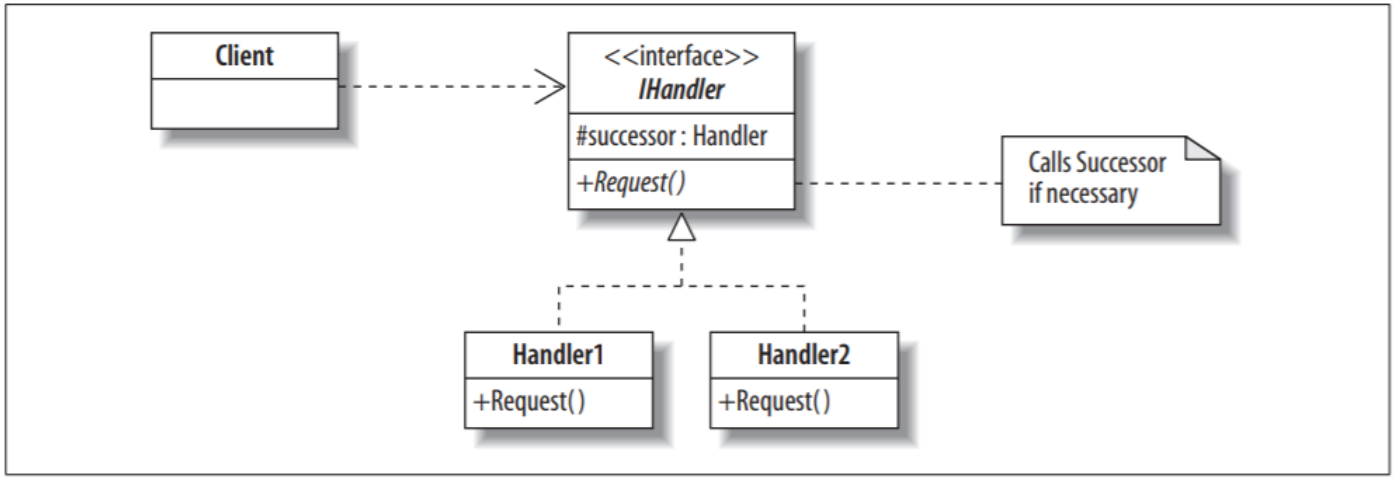
TEMPLATE

The Template Method pattern enables algorithms to defer certain steps to subclasses. The structure of the algorithm does not change, but small well-defined parts of its operation are handled elsewhere.



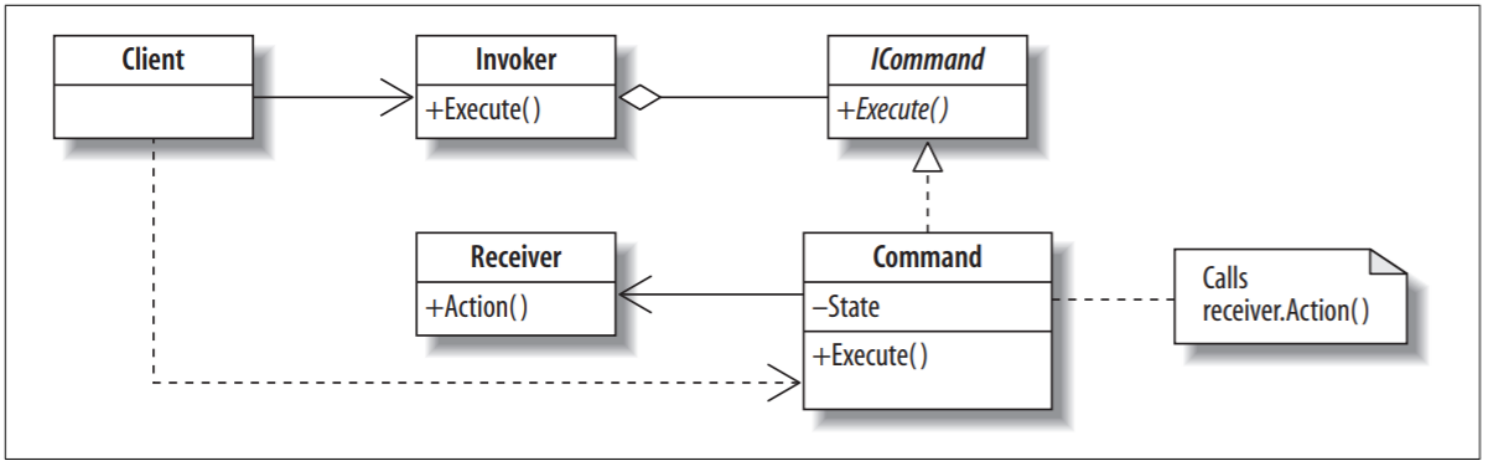
CHAIN OF RESPONCIBILITY

The Chain of Responsibility pattern works with a list of Handler objects that have limitations on the nature of the requests they can deal with. If an object cannot handle a request, it passes it on to the next object in the chain. At the end of the chain, there can be either default or exceptional behavior.



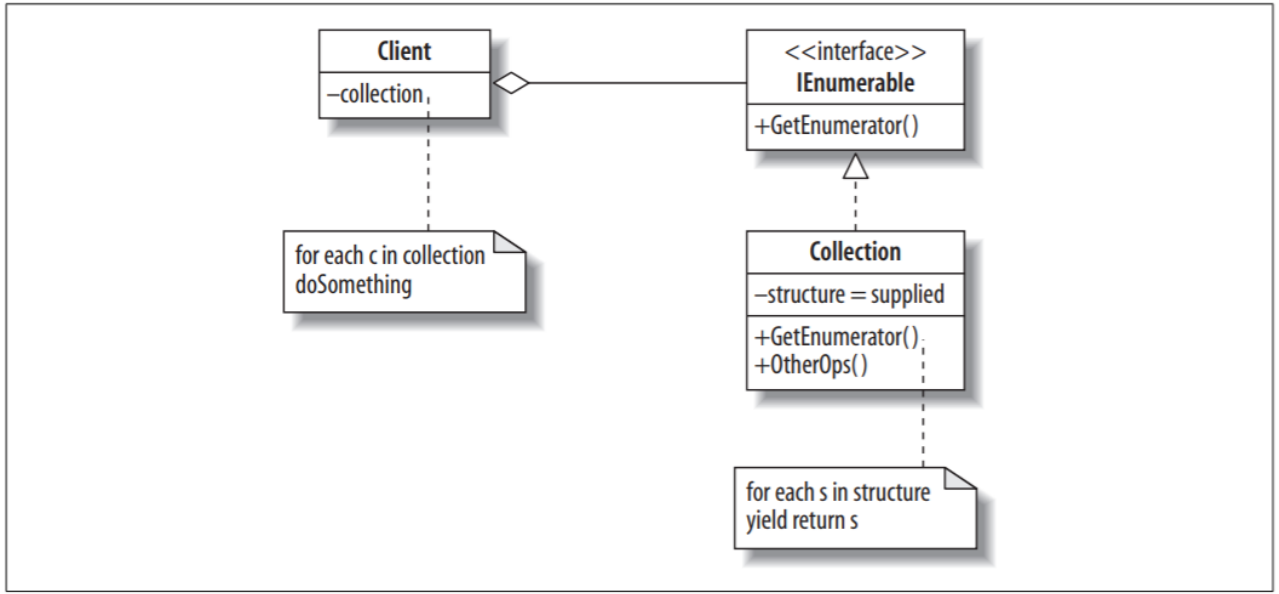
COMMAND

The Command pattern creates distance between the client that requests an operation and the object that can perform it.



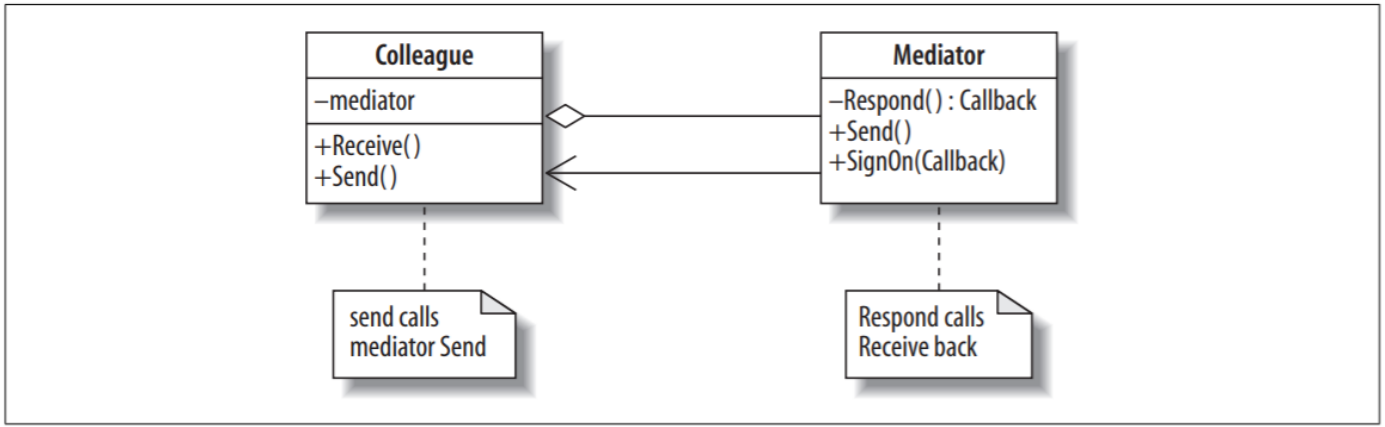
ITERATOR

The Iterator pattern provides a way of accessing elements of a collection sequentially, without knowing how the collection is structured. As an extension, the pattern allows for filtering elements in a variety of ways as they are generated.



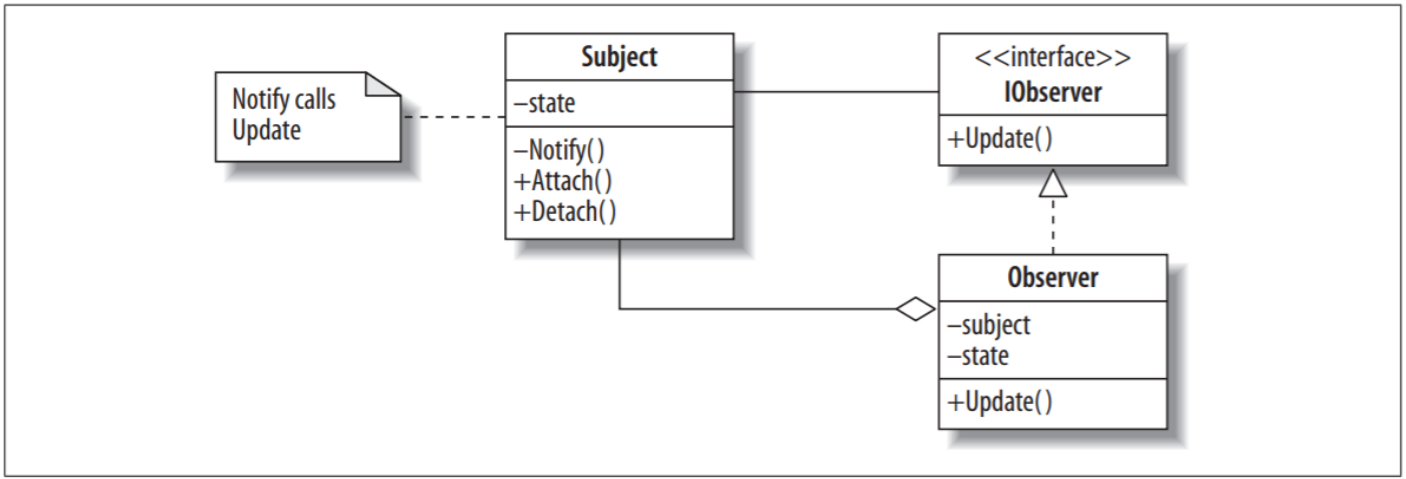
MEDIATOR

The Mediator pattern is there to enable objects to communicate without knowing each other's identities. It also encapsulates a protocol that objects can follow.



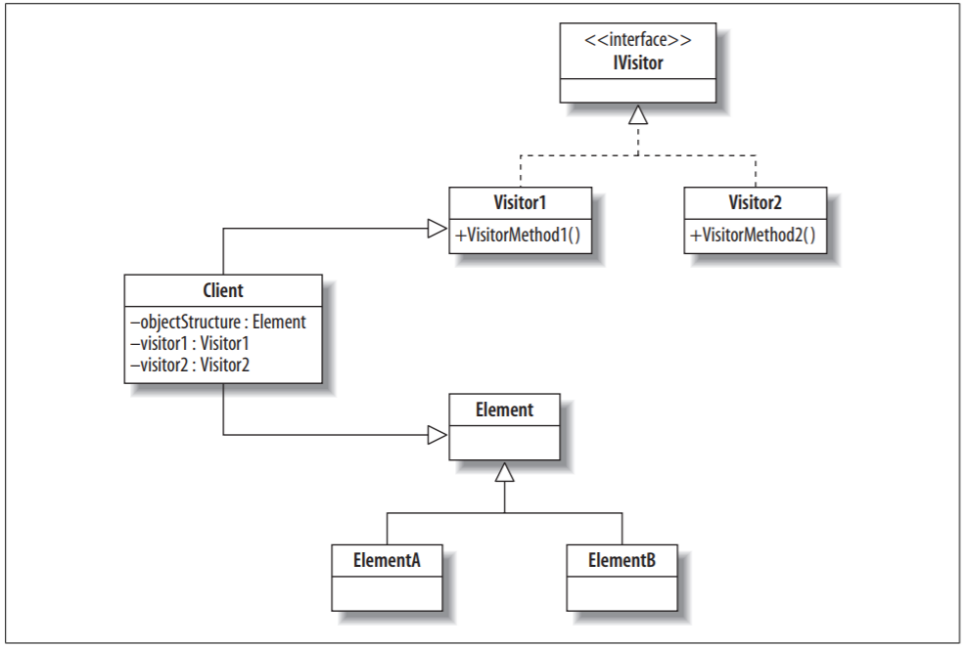
OBSERVER

The Observer pattern defines a relationship between objects so that when one changes its state, all the others are notified accordingly. There is usually an identifiable single publisher of new state, and many subscribers who wish to receive it.



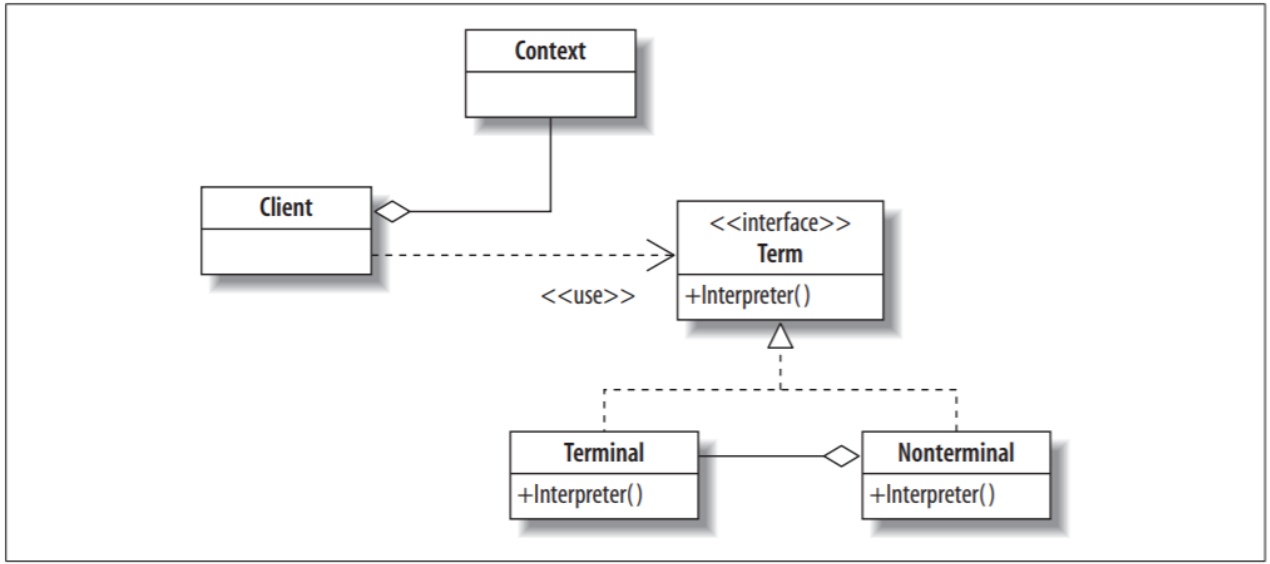
VISITOR

The Visitor pattern defines and performs new operations on all the elements of an existing structure, without altering its classes.



INTERPRETER

The Interpreter pattern supports the interpretation of instructions written in a language or notation defined for a specific purpose. The notation is precise and can be defined in terms of a grammar.



MEMENTO

This pattern is used to capture an object's internal state and save it externally so that it can be restored later.