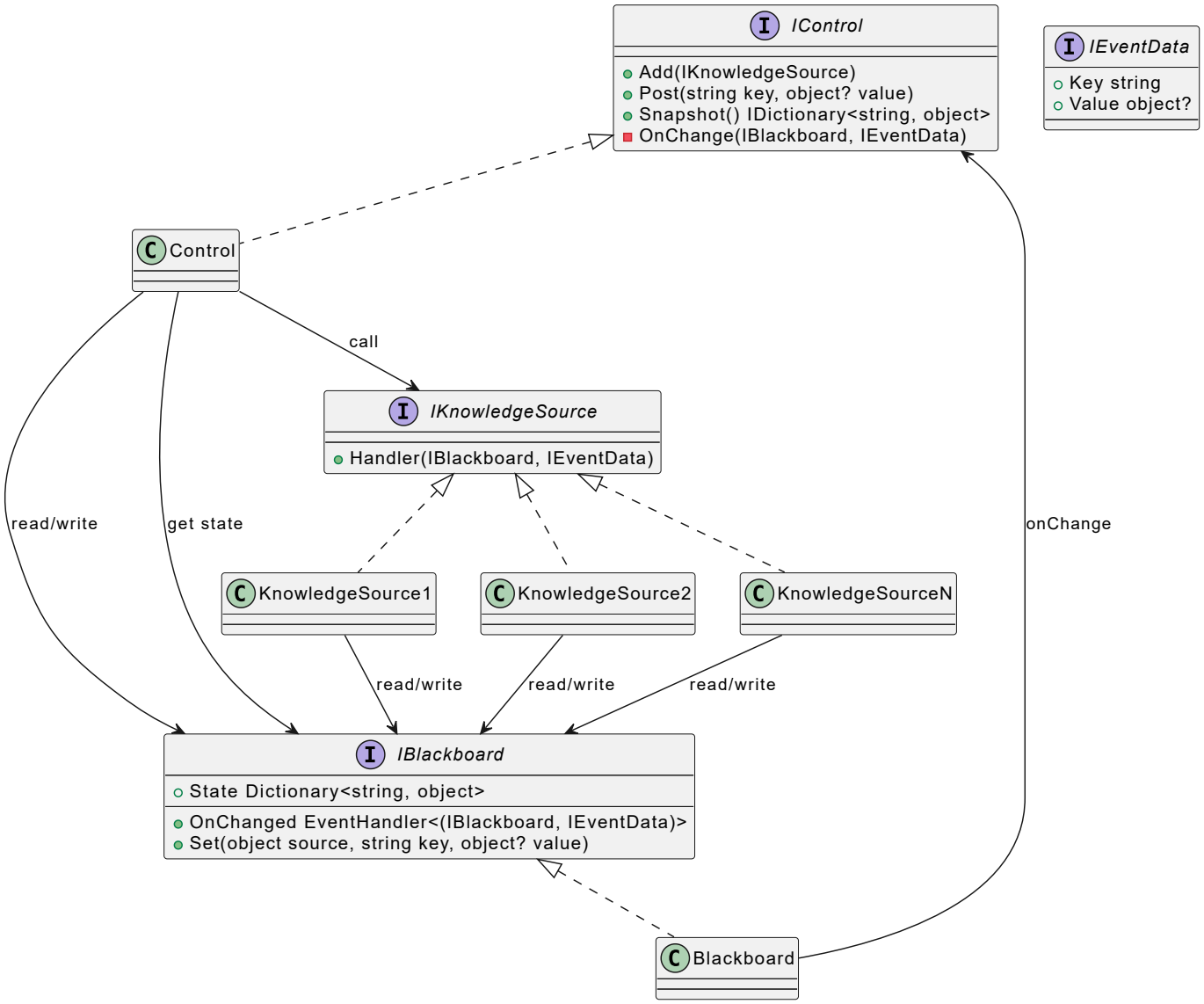
Behavioral Patterns

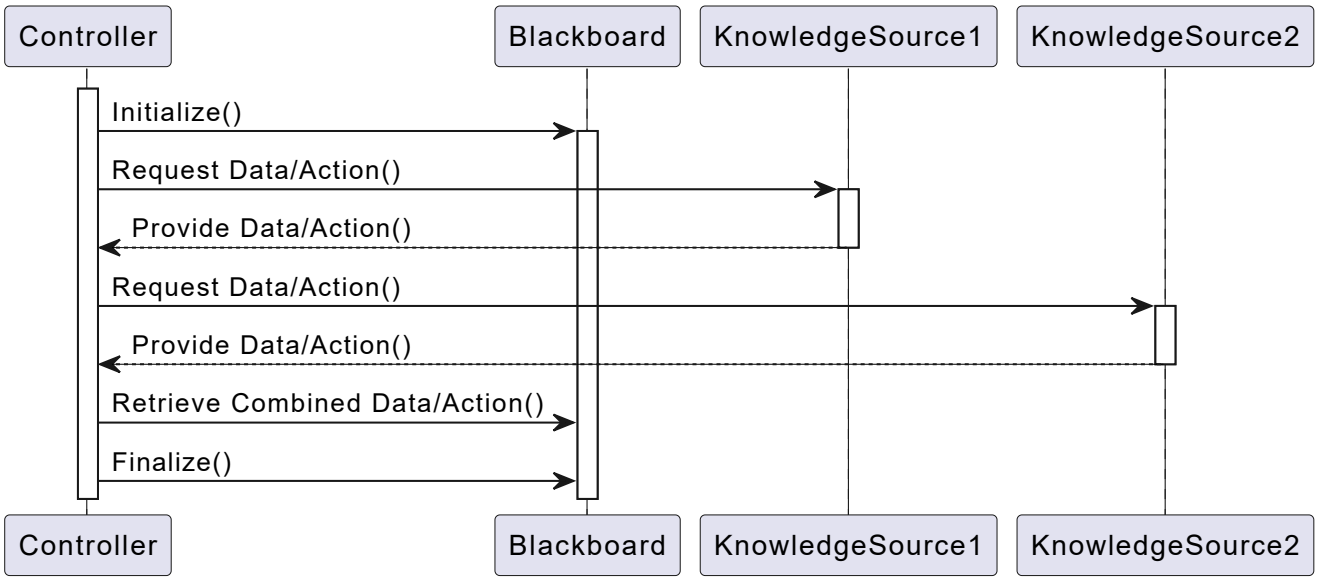
Blackboard

The blackboard pattern allows discrete state management across multiple knowledge source providers. Each knowledge provider owns its own validate and data lookup. Shared state is stored in the Blackboard and access in managed through the control.

Class Diagram



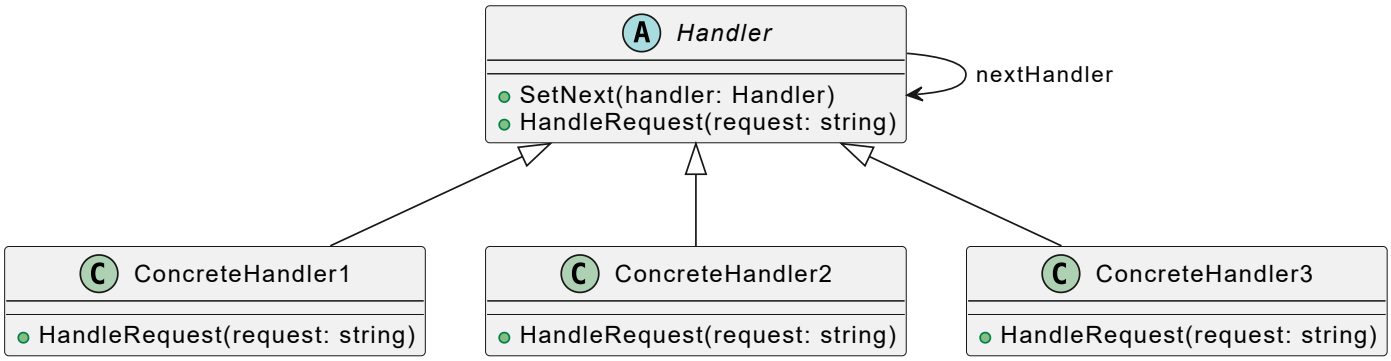
Sequence Diagram



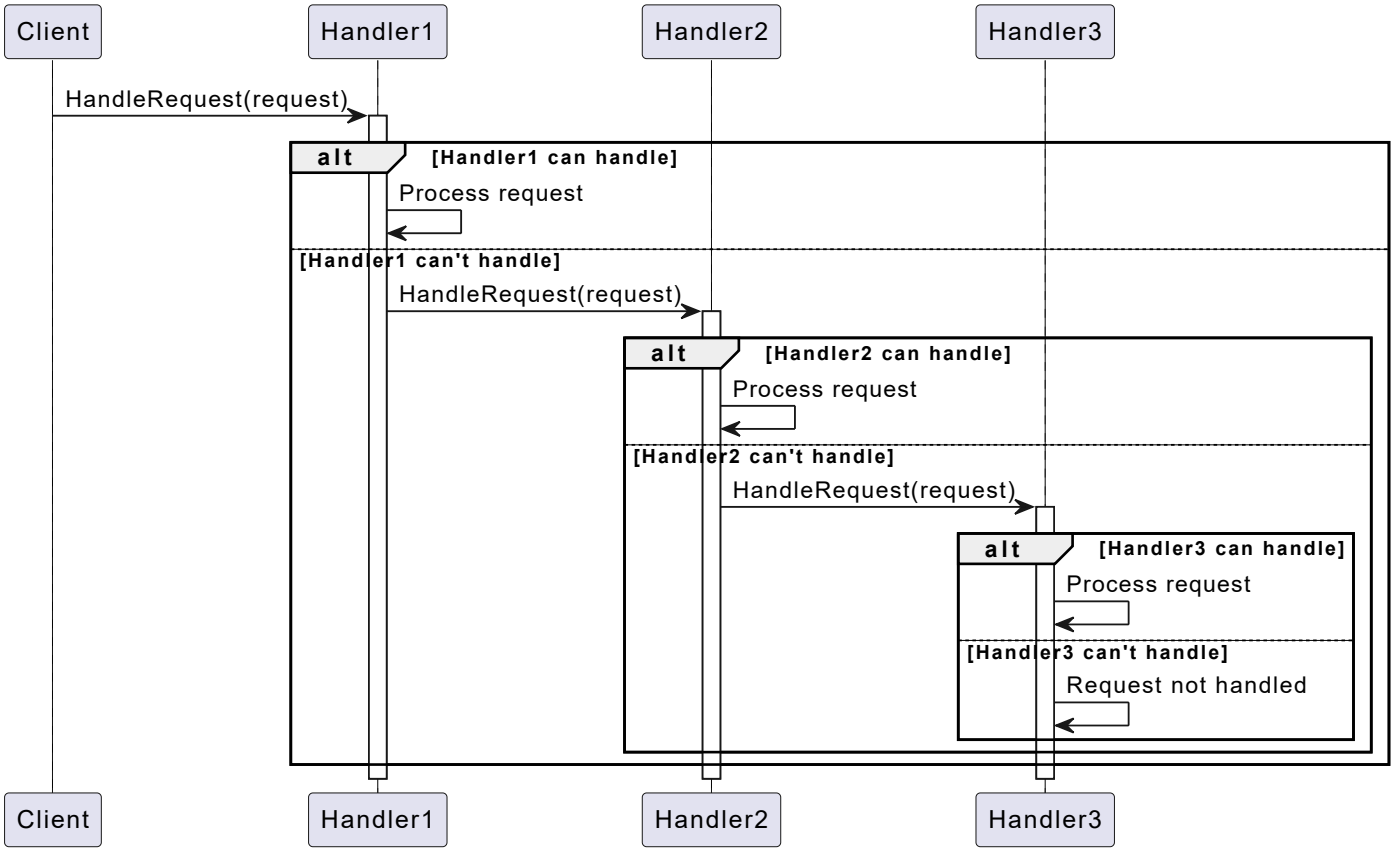
Chain of Responsibility

Chain of Responsibility provides a means to decouple the requester from the actual handler. If the called operation is not able to complete the request it is passed on to the next handler in the chain until the chain ends or the request is handled.

Class Diagram



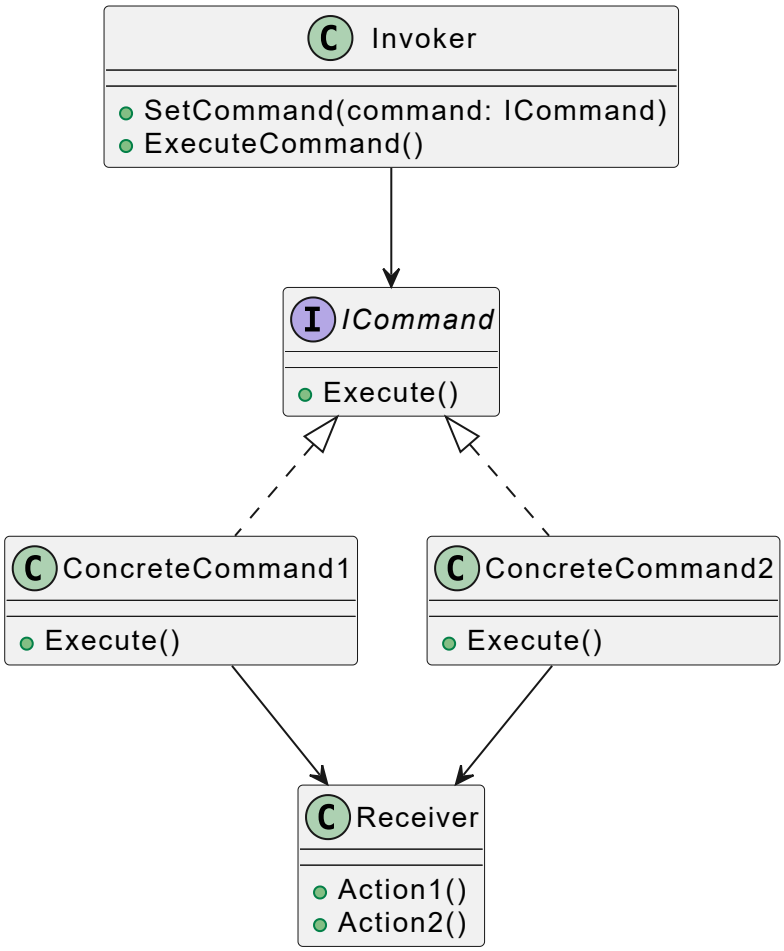
Sequence Diagram



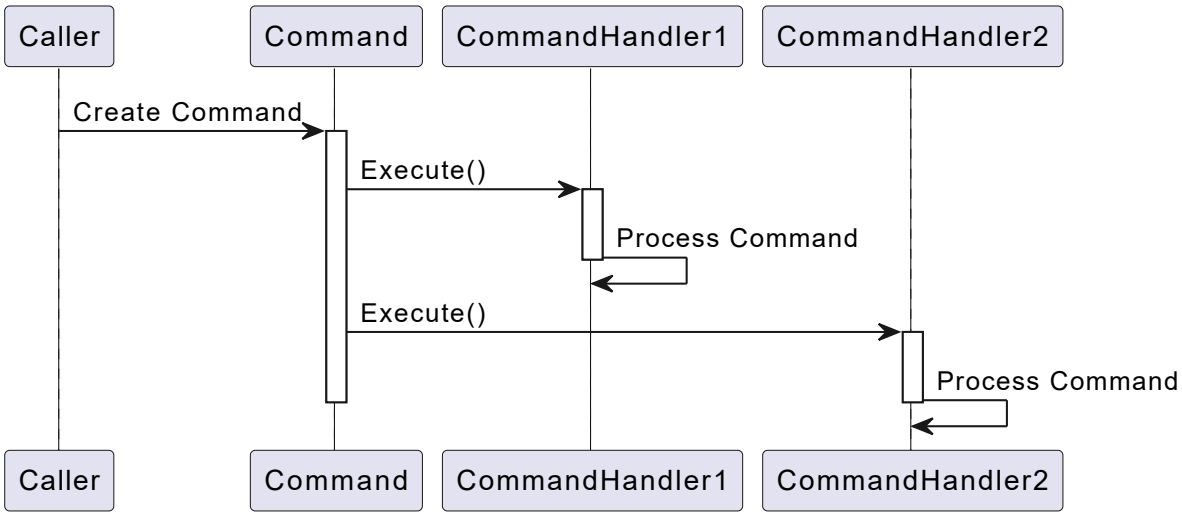
Command

The command pattern allows for defining a common means to execute operations. This allows for decoupling the caller from the handler.

Class Diagram



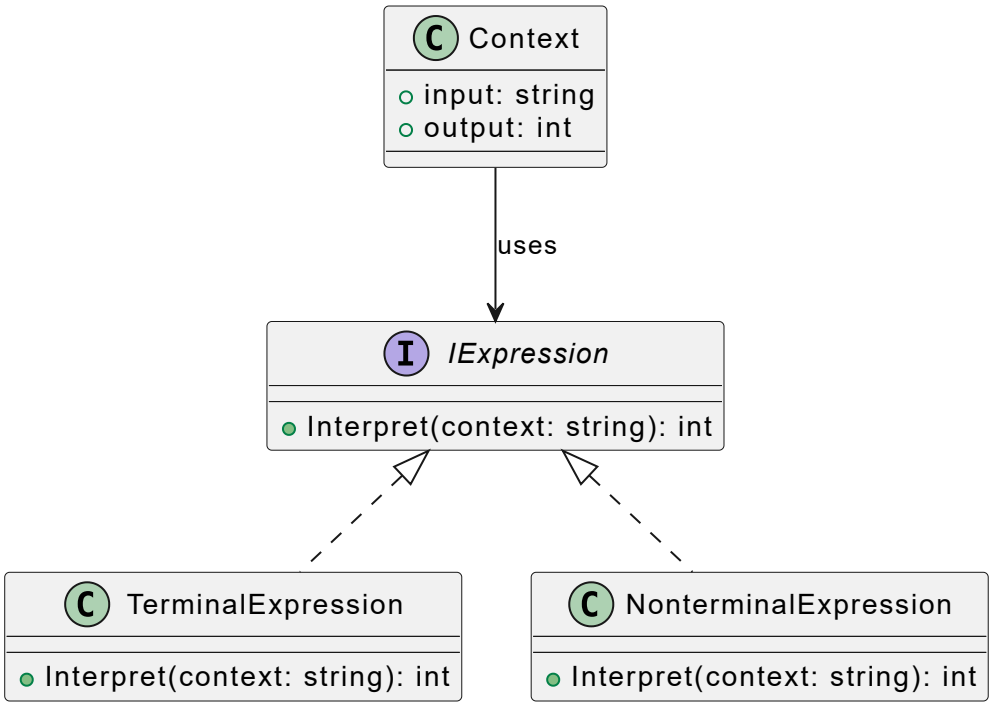
Sequence Diagram



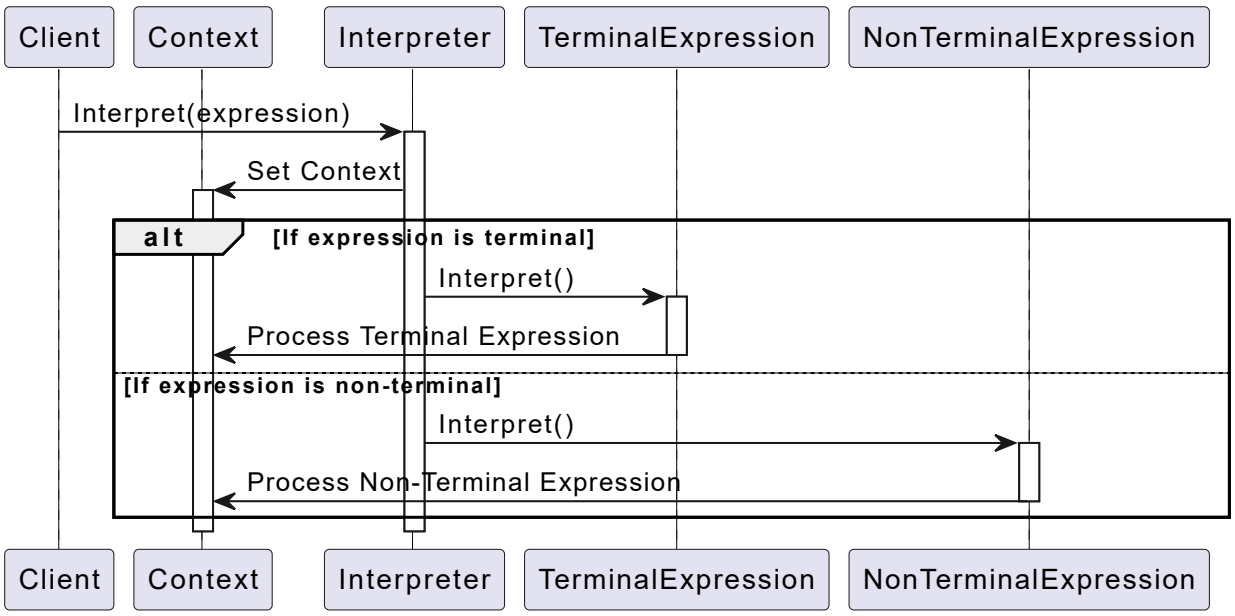
Interpreter

Interpreters are used to declare a graph based representation of a syntax expression such as the abstract syntax tree (AST) behind a expression parser.

Class Diagram



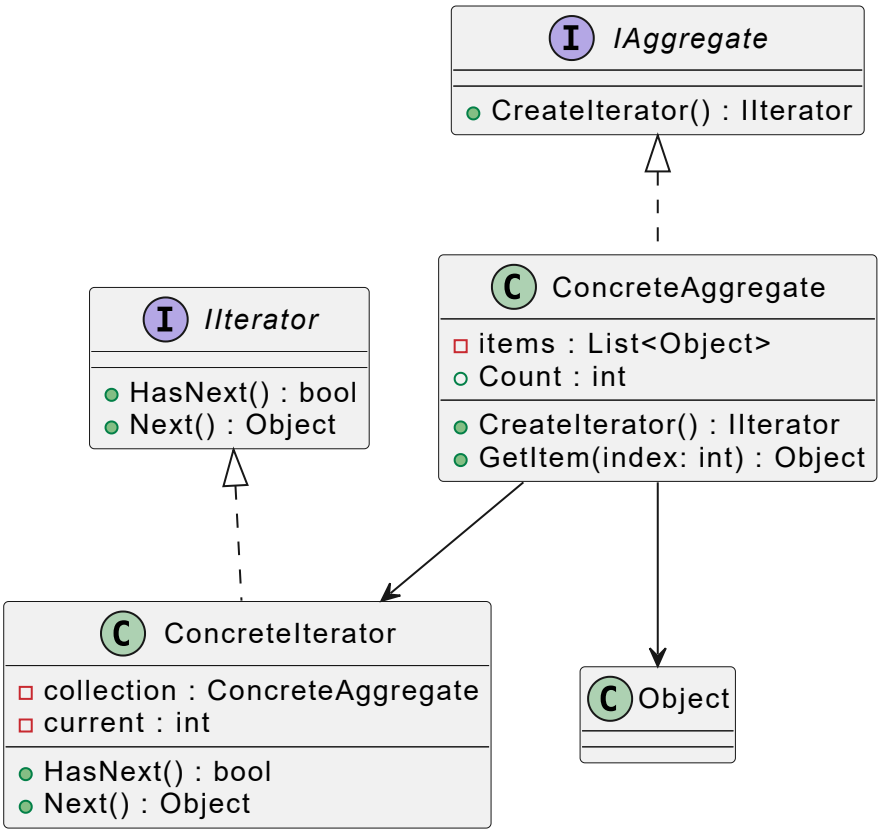
Sequence Diagram



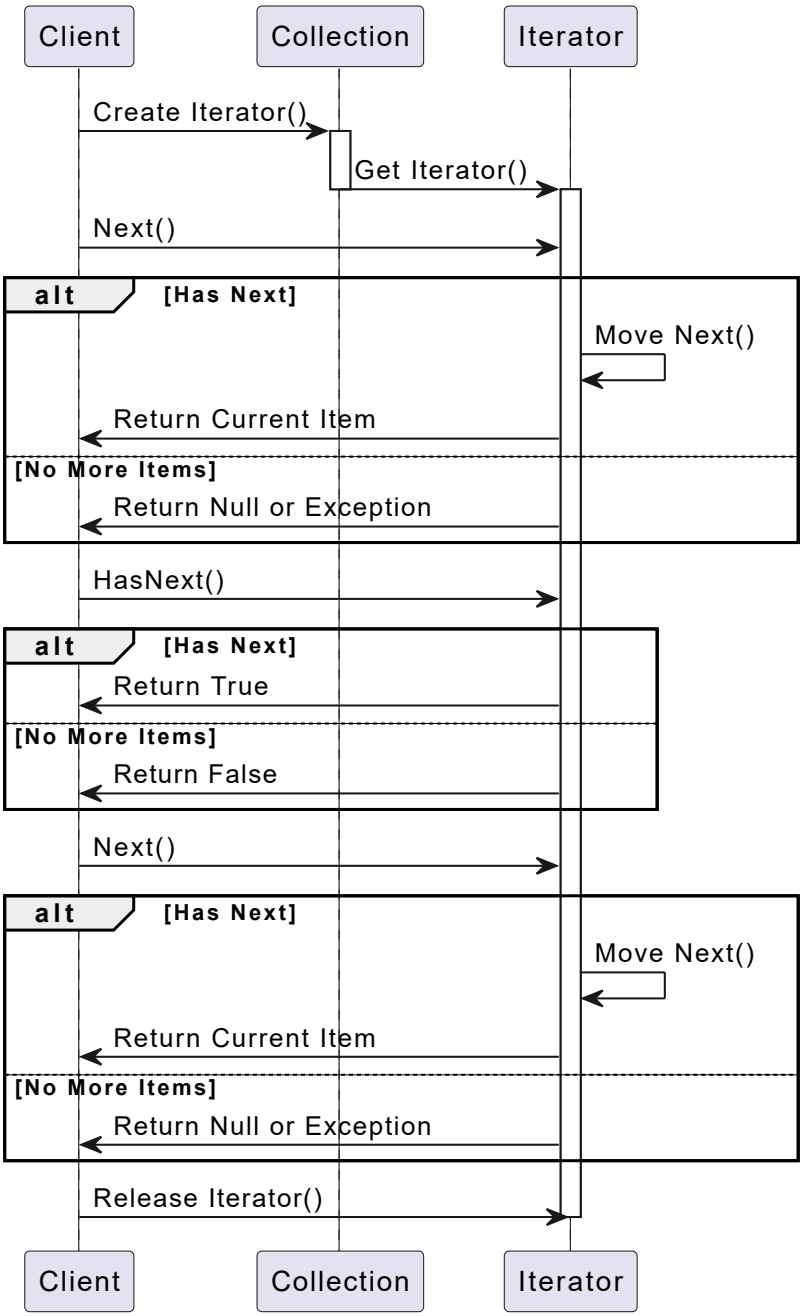
Iterator or Cursor

Iterators are used to access the elements of a set of objects sequentially.

Class Diagram

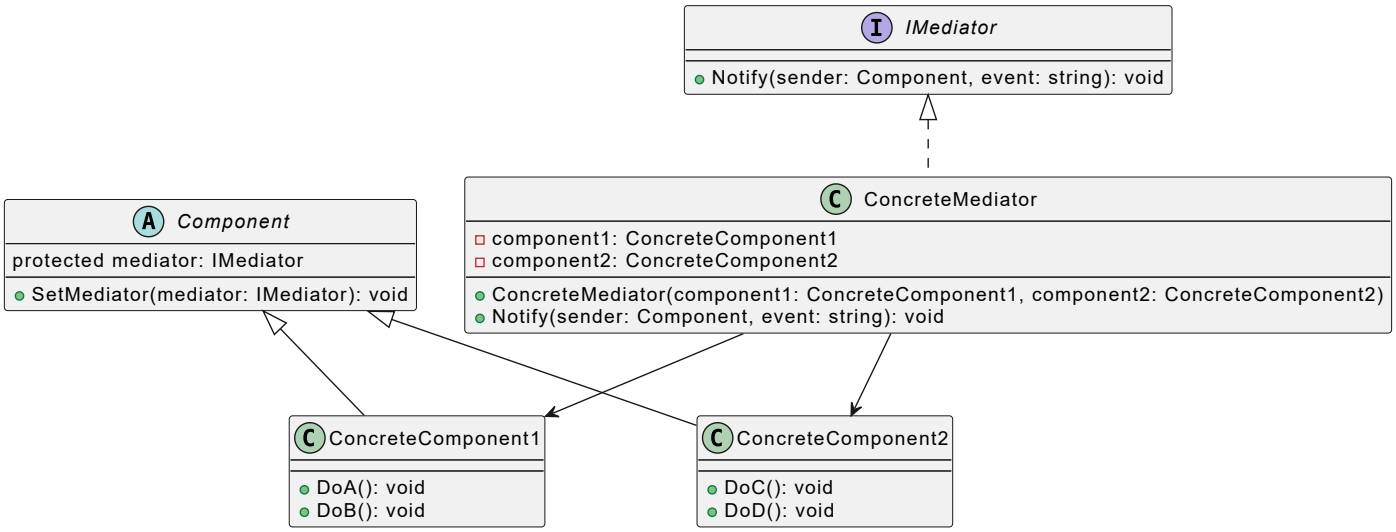


Sequence Diagram



Mediator

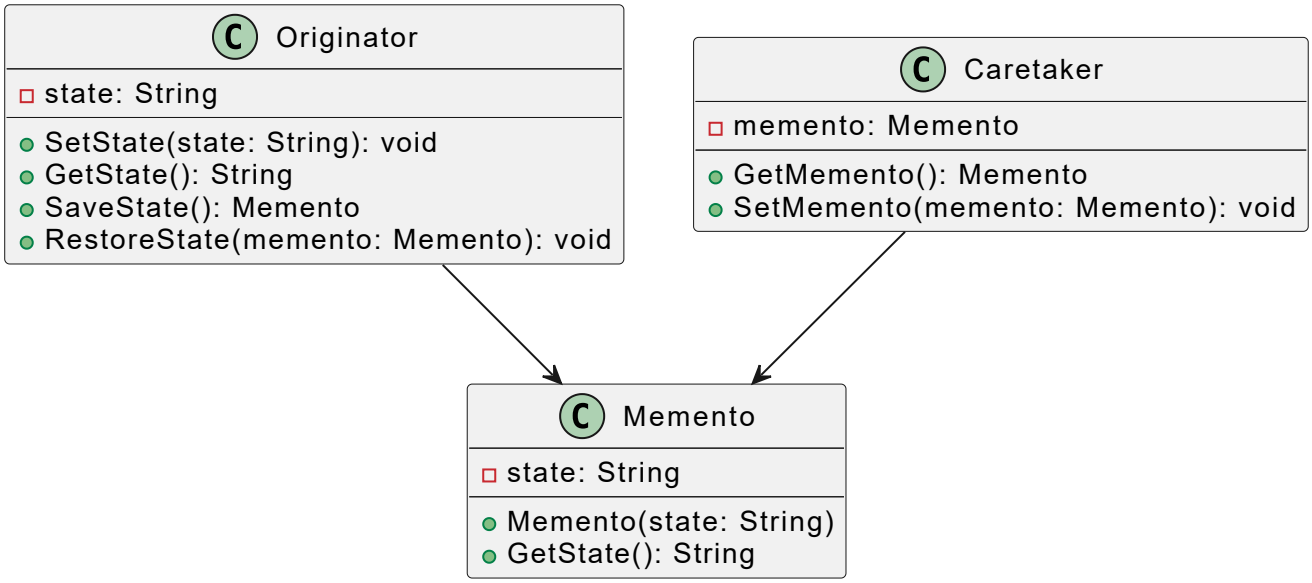
Mediator is used as a meant ot separate object interactions.



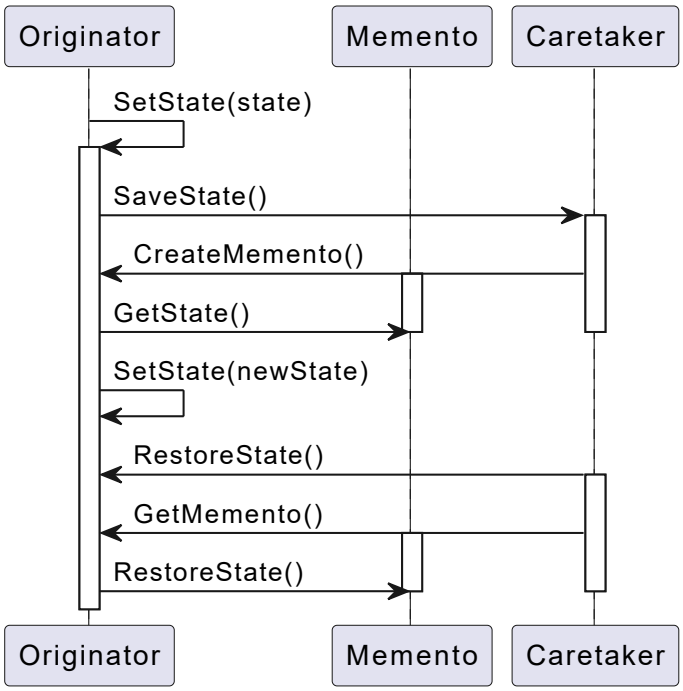
Memento

Memento is a mean to capture/replay state for an applications. Examples use cases may include undo/redo functionality or change a change log.

Class Diagram



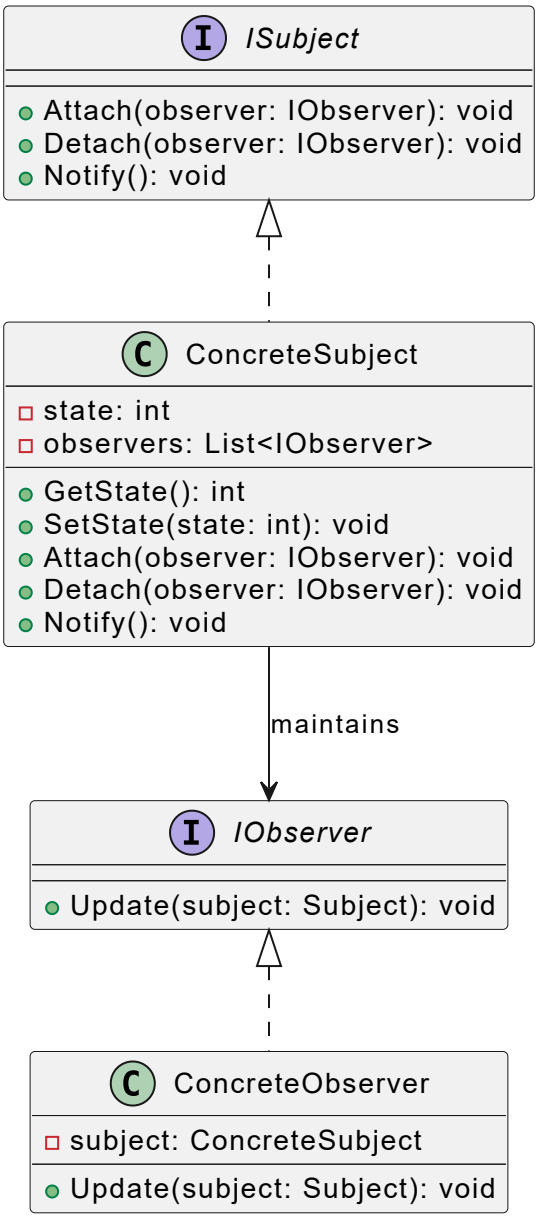
Sequence Diagram



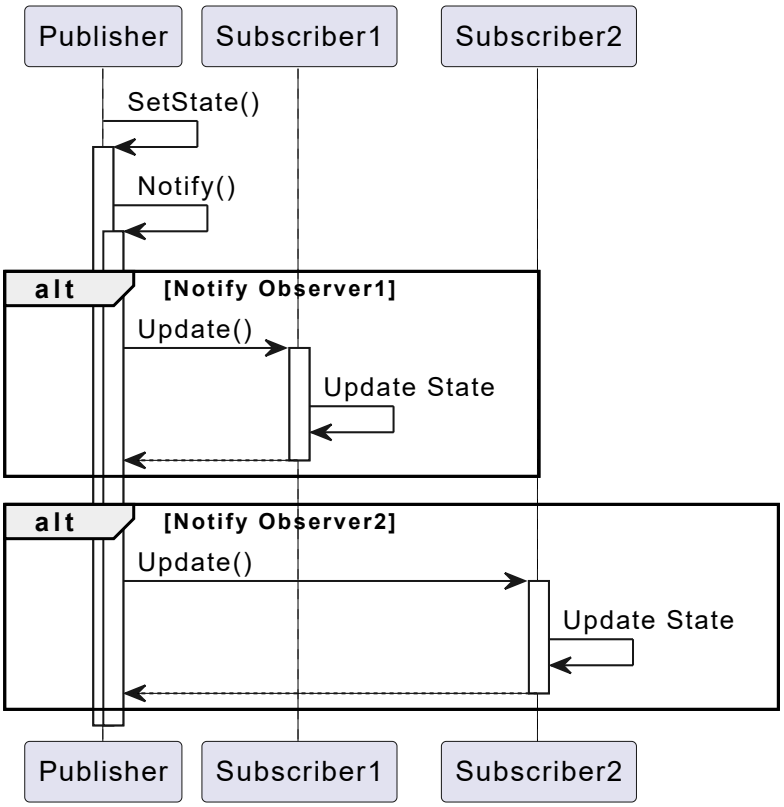
Observer, Dependents or Publisher/Subscriber

Observers are used to track changes and notify dependents.

Class Diagram



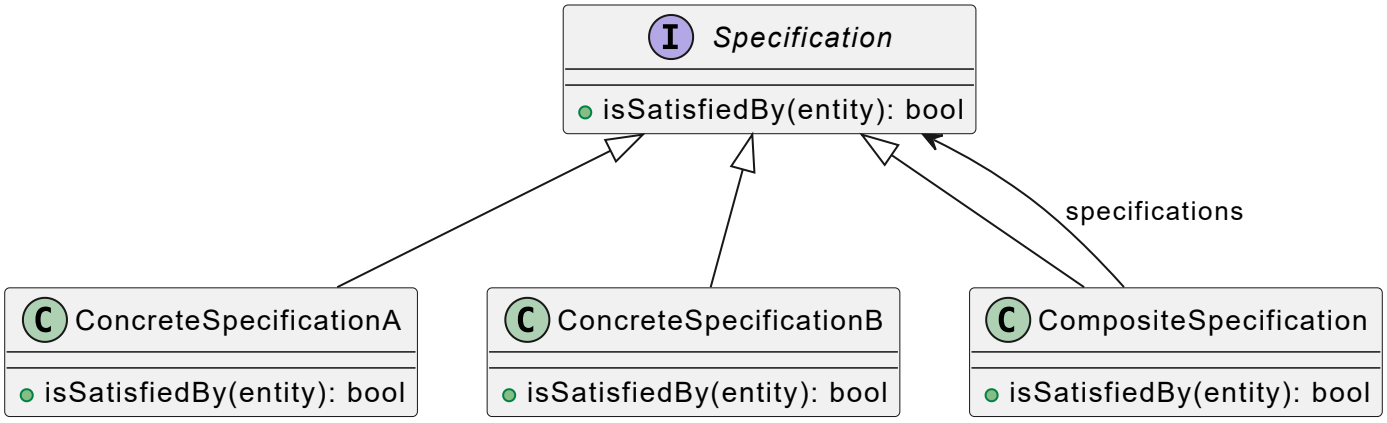
Sequence Diagram



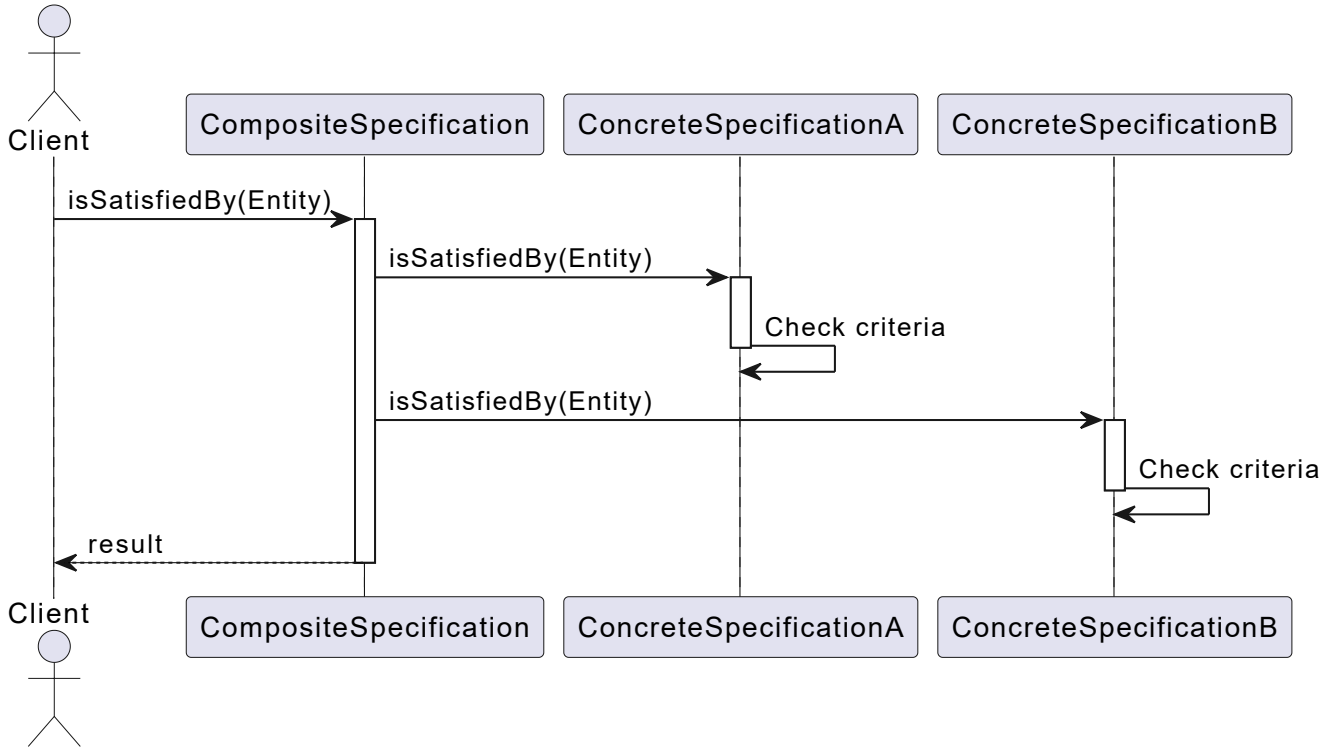
Specification

The Specifications pattern is used as a means to define combinable business logic

Class Diagram

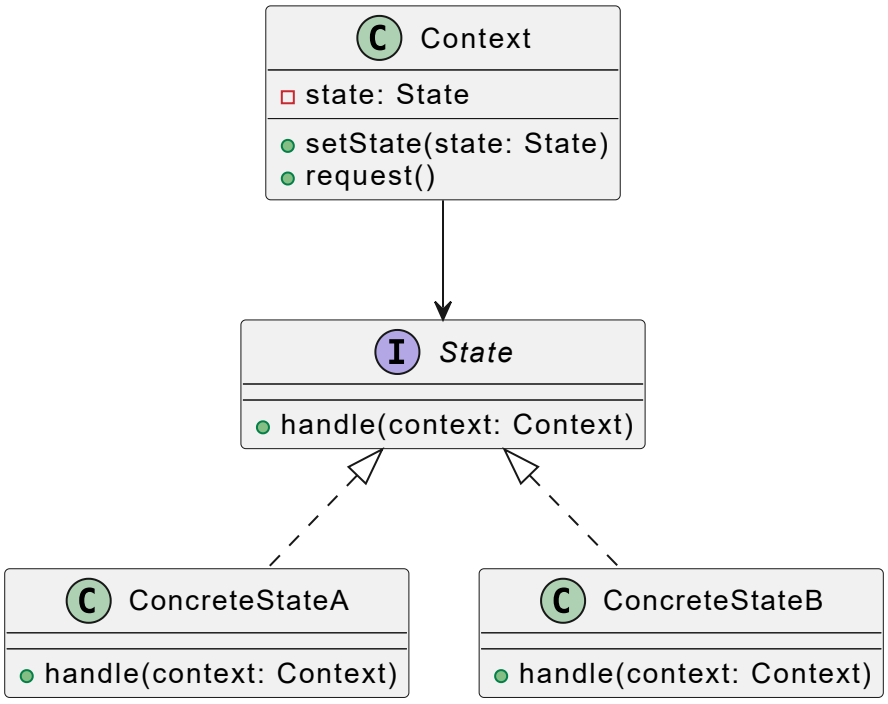


Sequence Diagram

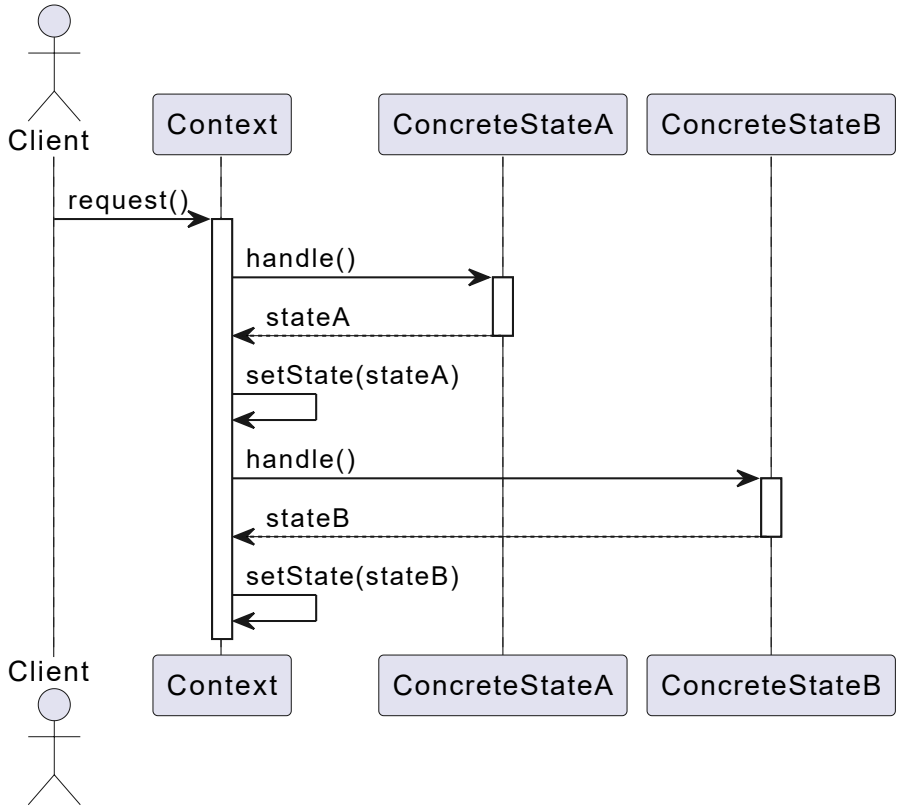


State

Class Diagram



Sequence Diagram


Client
Context
ConcreteStateA
ConcreteStateB
request()
handle()
stateA
setState(stateA)
handle()
stateB
setState(stateB)
Client
Context
ConcreteStateA
ConcreteStateB