Design Patterns

Lecture 2

Manuel Mastrofini

Systems Engineering and Web Services

University of Rome Tor Vergata

June 2011

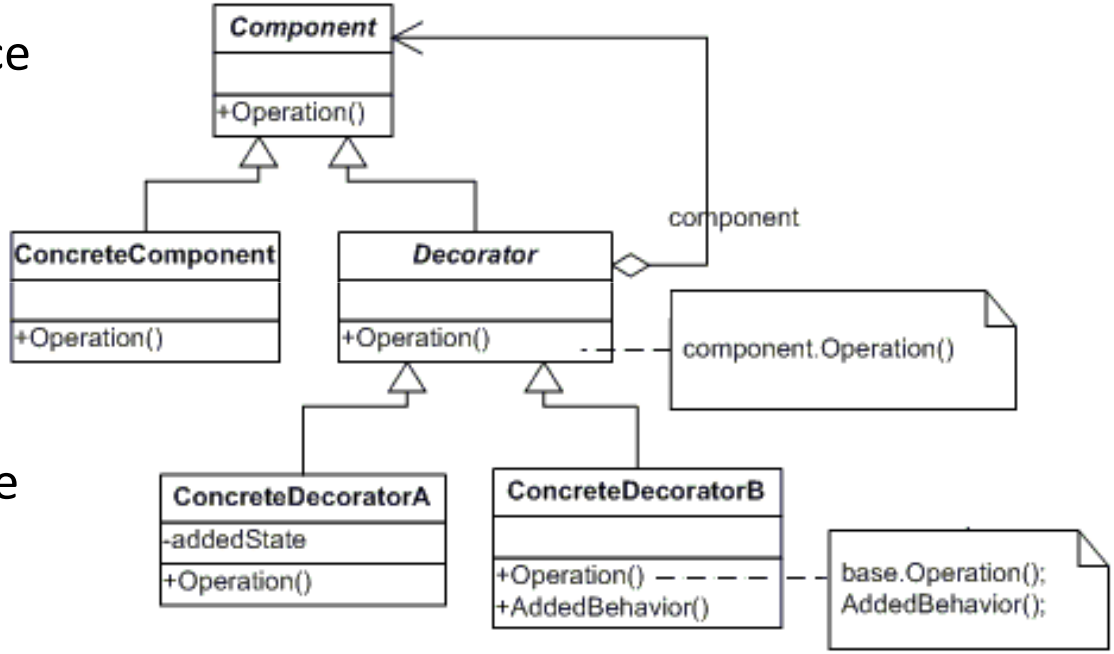
Structural patterns

Part 2

Decorator

- **Intent**: It attaches additional responsibilities to an object dynamically.

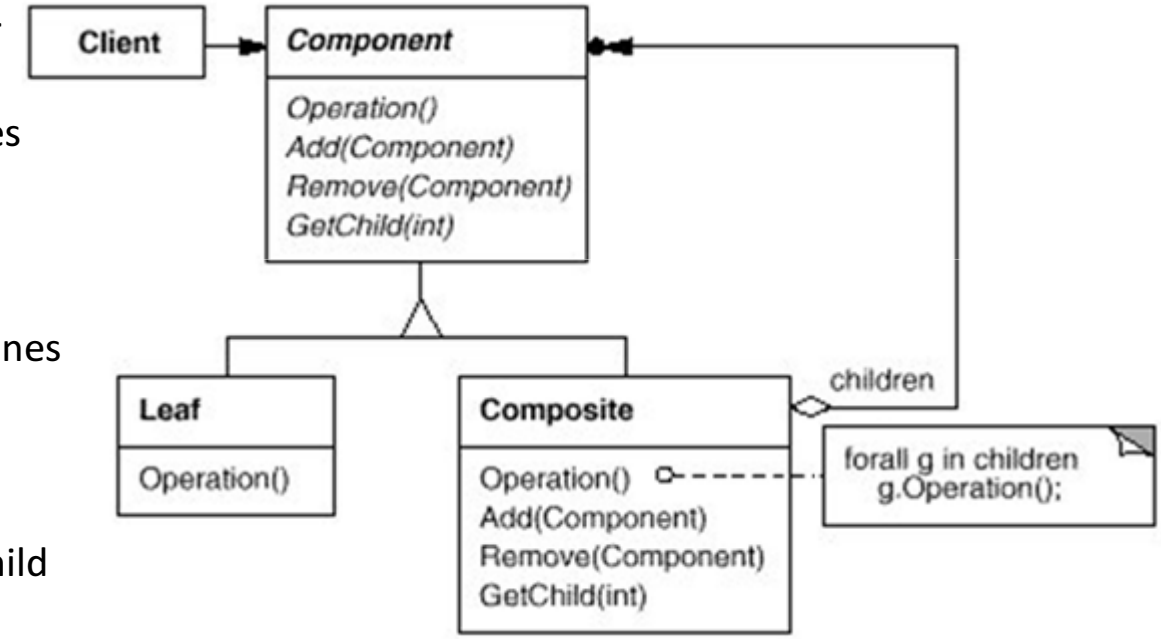
- ***Component***: defines the interface for objects that can have responsibilities added to them dynamically.
- ***ConcreteComponent***: defines an object to which additional responsibilities can be attached.
- ***Decorator***: maintains a reference to a Component object and defines an interface that conforms to Component's interface.
- ***ConcreteDecorator***: adds responsibilities to the component.



Composite

- **Intent**: It composes objects into tree structures to represent part-whole hierarchies, letting clients treat individual objects and compositions of objects uniformly.

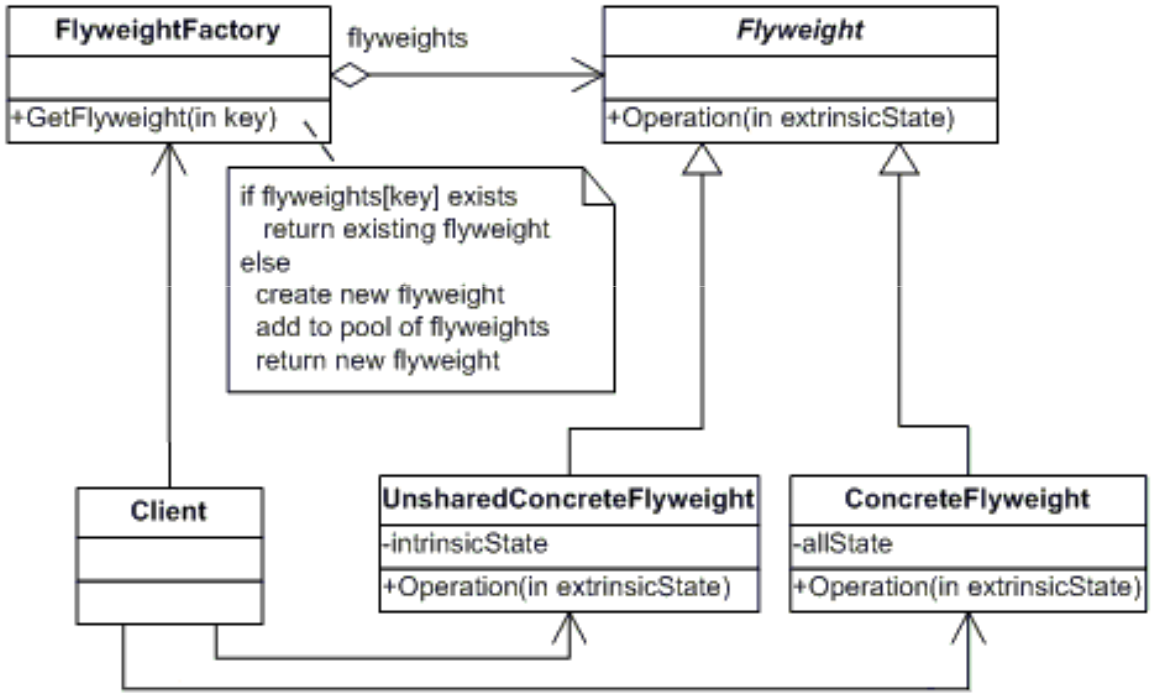
- ***Component***: declares the interface for objects in the composition and implements default behaviors; declares an interface for managing its child components.
- ***Leaf***: represents leaf objects in the composition, has no children, and defines behavior for primitive objects in the composition.
- ***Composite***: defines behavior for components having children, stores child components, and implements child-related operations in the Component interface.



Variant: Caching can be used for improving performance, but children should be allowed to invalidate cache.
Variant: Children can keep a reference to their parents (see Chain of Responsibility pattern).

Flyweight

- **Intent**: It uses sharing to support large numbers of fine-grained objects efficiently.

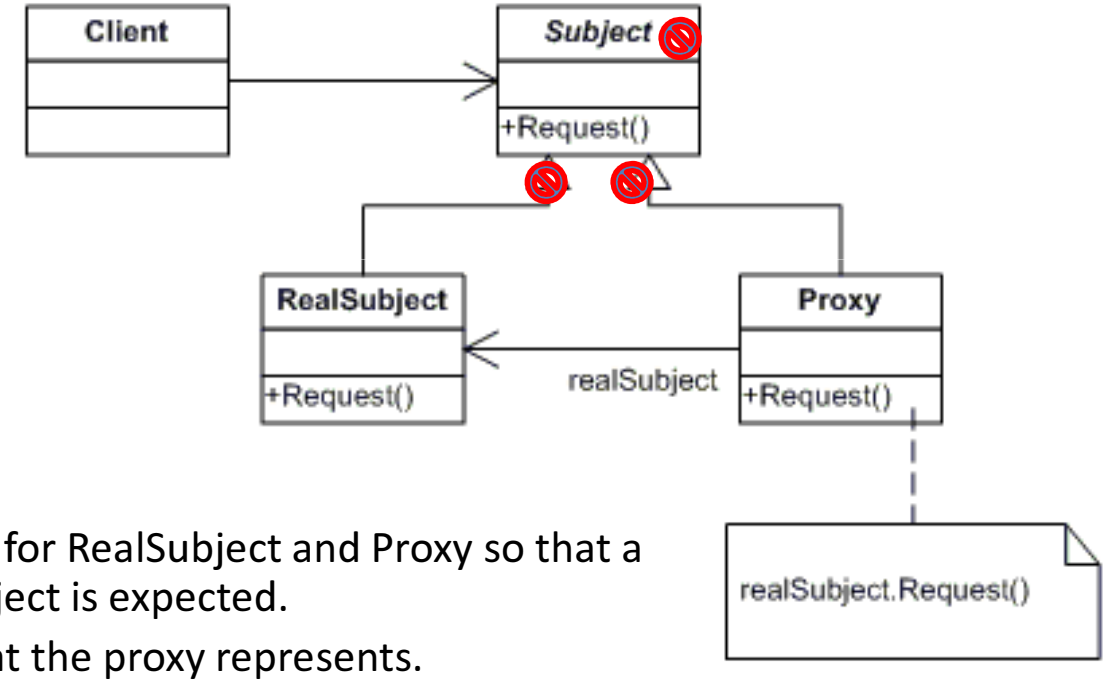


- ***Flyweight*** : an interface through which flyweights can receive and act on extrinsic state.
- ***ConcreteFlyweight***: implements the Flyweight interface and adds storage for intrinsic state, if any. Any state it stores must be intrinsic; that is, it must be independent of the object's context.
- ***FlyweightFactory***: creates and manages flyweight objects. It ensures that flyweights are shared properly. When a client requests a flyweight, it supplies an existing instance or creates one, if none exists.
- ***UnsharedConcreteFlyweight***: not all Flyweight subclasses need to be shared. The Flyweight interface enables sharing; it doesn't enforce it. It's common for UnsharedConcreteFlyweight objects to have ConcreteFlyweight objects as children at some level in the flyweight object structure.

Proxy (Surrogate)

- **Intent**: It provides a surrogate or placeholder for another object to control access to it.

- ***Proxy***: maintains a reference that lets the proxy access the real subject. Proxy may refer to a Subject if the RealSubject and Subject interfaces are the same. Furthermore, it provides an interface identical to Subject's so that a proxy can be substituted for the real subject and it controls access to the real subject and may be responsible for creating and deleting it.



- ***Subject***: defines the common interface for RealSubject and Proxy so that a Proxy can be used anywhere a RealSubject is expected.
- ***RealSubject***: defines the real object that the proxy represents.

Variant: Direct access to RealSubject from Client can be disabled for security or performance reasons, but also for hiding the physical localization of RealSubject.

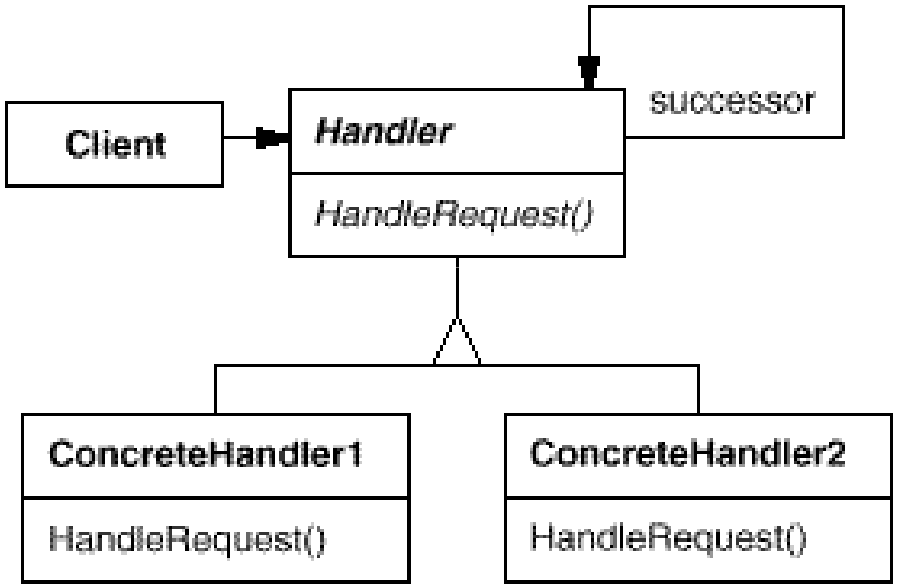
Variant: Proxy can perform additional operations after/before the request.

Behavioral patterns

Chain of Responsibility

- **Intent**: It avoids coupling the sender of a request to its receiver by giving more than one object a chance to handle the request.

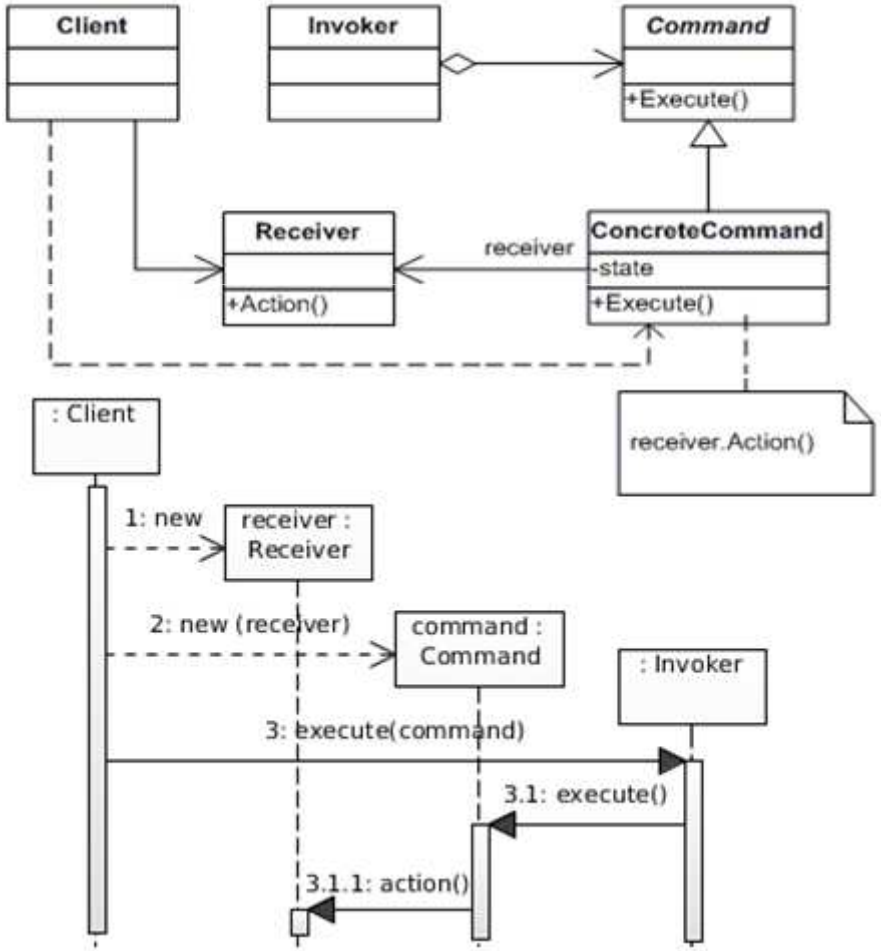
- ***Handler***: defines an interface for handling requests.
- ***ConcreteHandler***: handles requests it is responsible for, can access its successor, and if the ConcreteHandler can handle the request, it does so; otherwise it forwards the request to its successor.



Command (Action, Transaction)

- **Intent**: It encapsulates a request as an object.

- ***Command***: declares an interface for executing an operation.
- ***ConcreteCommand***: defines a binding between a Receiver object and an action and implements Execute by invoking the corresponding operation(s) on Receiver.
- ***Invoker***: asks the command to carry out the request.
- ***Receiver***: knows how to perform the operations associated with carrying out a request.

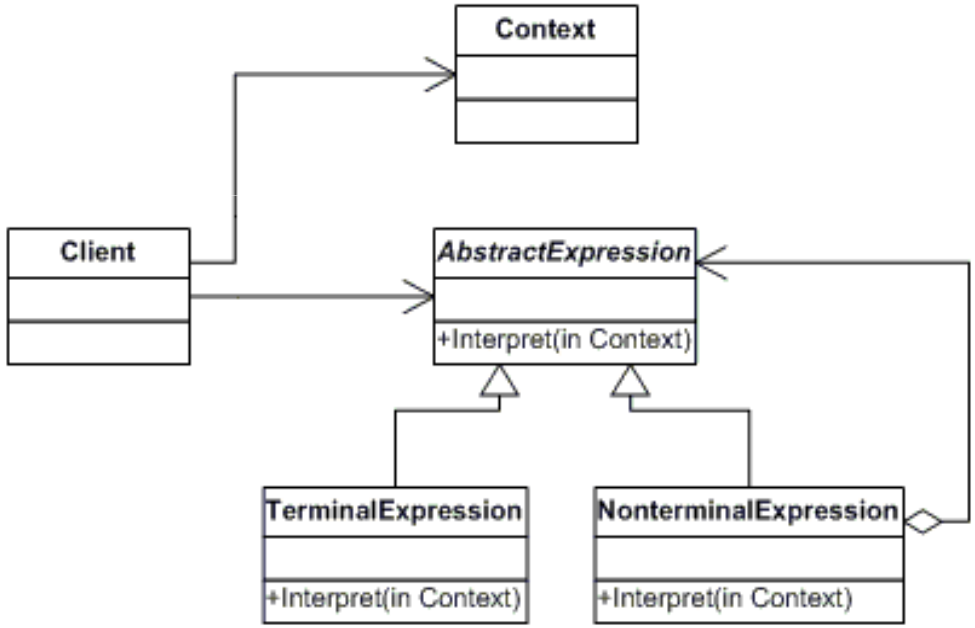


Variant: When Command leverages the Composite pattern, complex commands can be built.

Interpreter

- **Intent**: It defines a representation for the grammar of a language along with an interpreter that uses the representation to interpret sentences in the language.

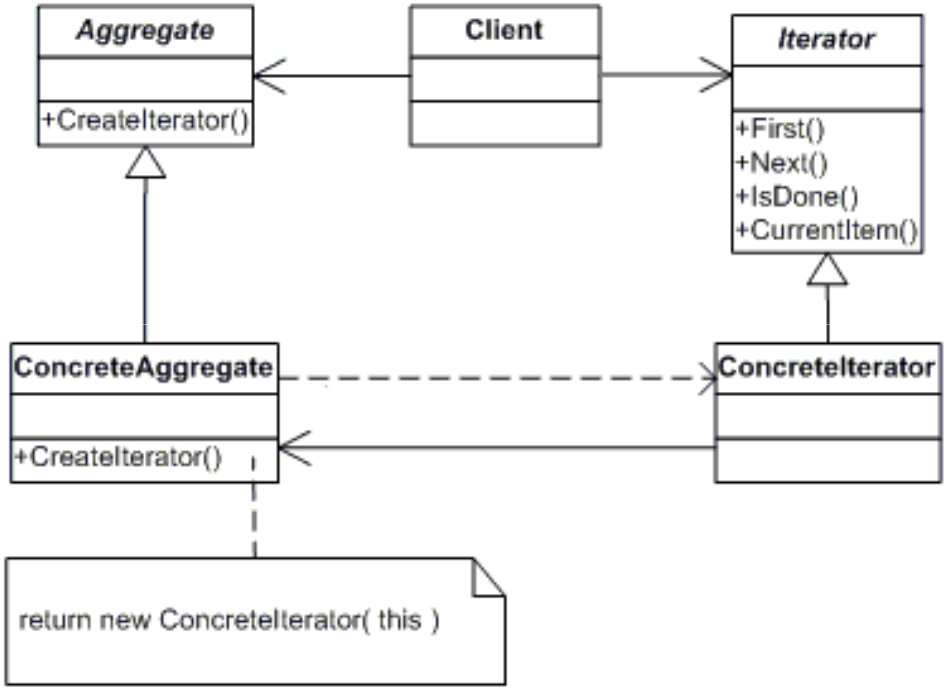
- ***AbstractExpression***: declares an abstract Interpret operation that is common to all nodes in the abstract syntax tree.
- ***TerminalExpression***: implements an Interpret operation associated with terminal symbols in the grammar. An instance is required for every terminal symbol in a sentence.
- ***NonterminalExpression***: one such class is required for every rule in the grammar and maintains instance variables of type AbstractExpression for each of the symbols. It implements an Interpret operation for nonterminal symbols in the grammar. Interpret typically calls itself recursively on the variables representing the symbols.
- ***Context*****:** contains information that is global to the interpreter.



Iterator

- **Intent**: It provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

- ***Iterator***: defines an interface for accessing and traversing elements.
- ***ConcreteIterator***: implements the Iterator interface and keeps track of the current position in the traversal of the aggregate.
- ***Aggregate***: defines an interface for creating an Iterator object.
- ***ConcreteAggregate***: implements the Iterator creation interface to return an instance of the proper ConcreteIterator. A ConcreteIterator keeps track of the current object in the aggregate and can compute the succeeding object in the traversal



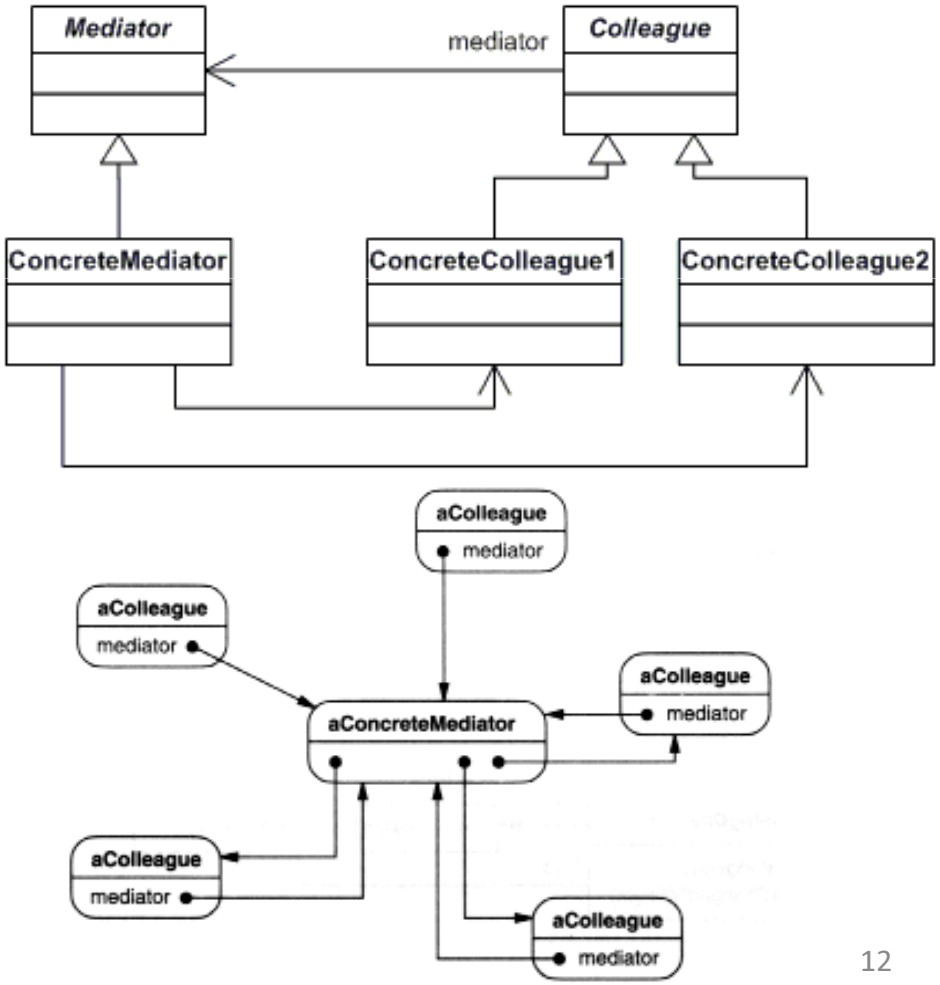
Variant: Clients explicitly ask for advancing the traversal and getting the next element (External iterator).
Variant: Clients ask for an operation which the Iterator performs on every element (Internal iterator).

Variant: The traversing algorithm may be defined in the Aggregate and the Iterator, which becomes a Cursor, just stores the information on the current element.

Mediator

- **Intent**: It defines an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

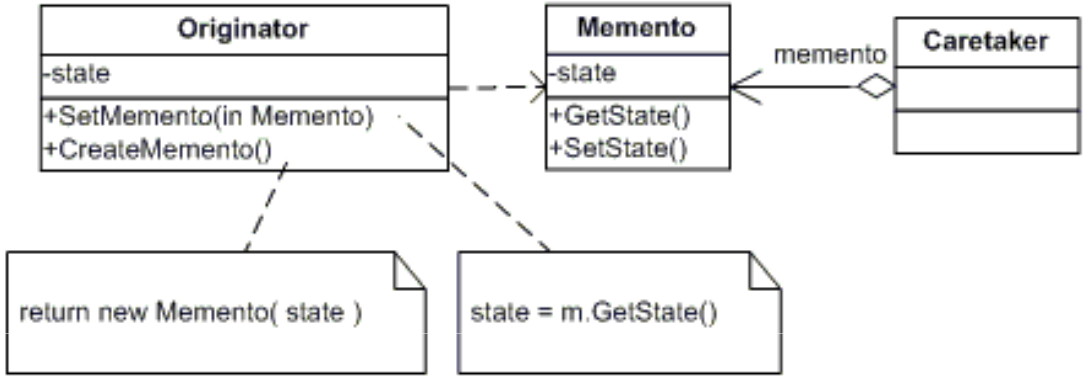
- ***Mediator***: defines an interface for communicating with Colleague objects.
- ***MediatorConcrete***: implements cooperative behavior by coordinating Colleague objects and knows and maintains its colleagues.
- ***ConcreteColleague***: each Colleague class knows its Mediator object and communicates with it whenever it would have otherwise communicated with another colleague.
- ***Colleague***: defines an interface for Colleagues.

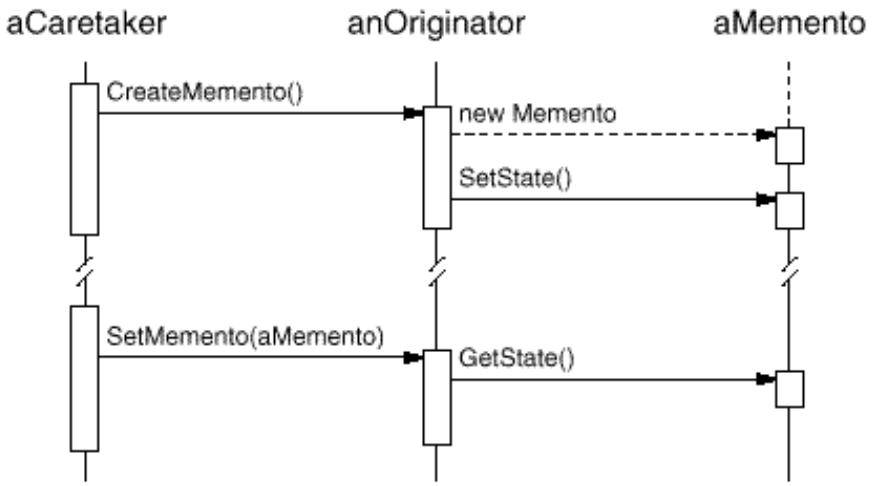


Memento (Token)

- **Intent**: It captures and externalizes an object's internal state so that the object can be restored to this state later.

- ***Memento***: stores internal state of the Originator object.
- ***Originator***: creates a memento containing a snapshot of its current internal state and uses the memento to restore its internal state.
- ***Caretaker***: is responsible for the memento's safekeeping and never operates on or examines the contents of a memento.



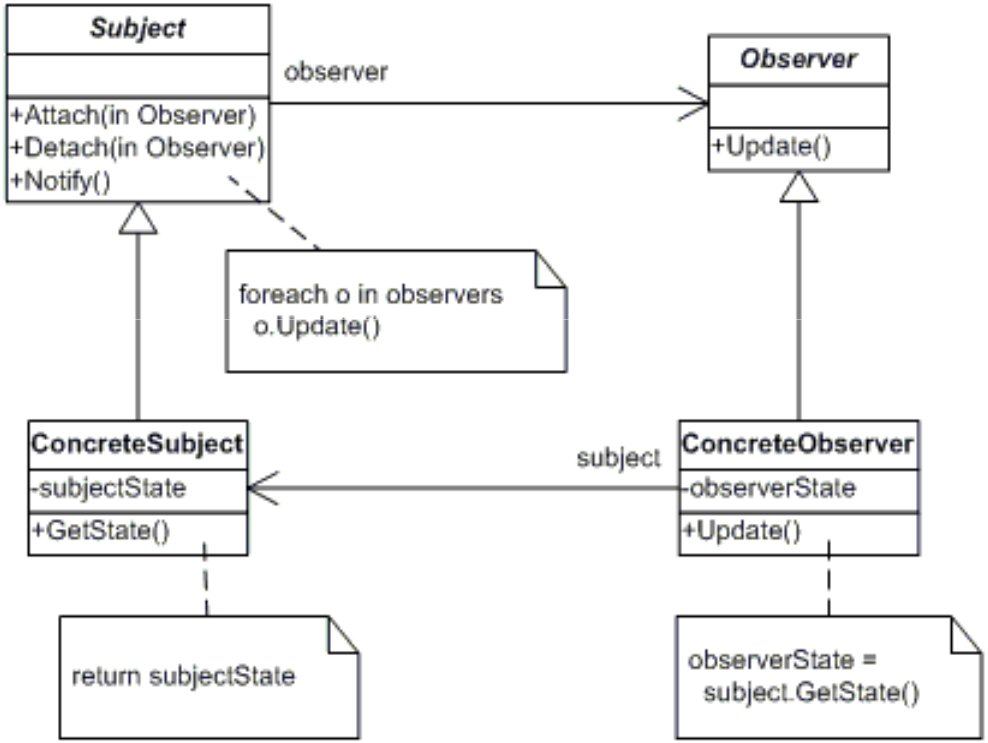


Variant: Instead of storing the entire state of the Originator, Memento may store just the incremental change to the Originator.

Observer (Dependents, Publish/Subscribe)

- **Intent**: it defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

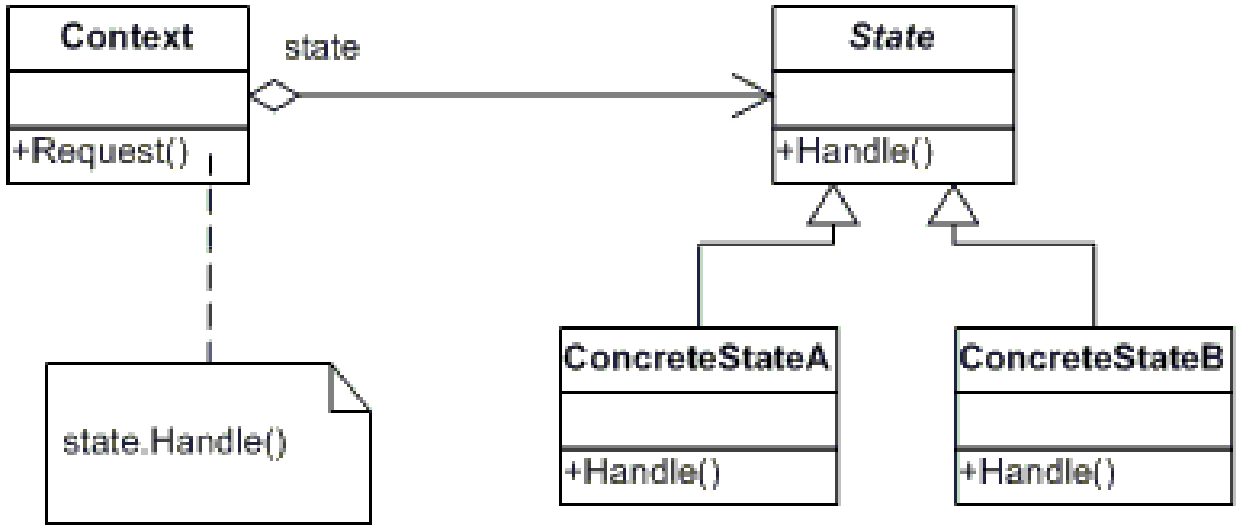
- ***Observer***: defines an updating interface for objects that should be notified of changes in a subject.
- ***ConcreteObserver***: maintains a reference to a ConcreteSubject object, stores state that should stay consistent with the subject's, and implements the Observer updating interface to keep its state consistent with the subject's
- ***Subject***: knows its observers. Any number of Observer objects may observe a subject. It provides an interface for attaching and detaching Observer objects.
- ***ConcreteSubject***: stores state of interest to ConcreteObserver objects and sends a notification to its observers when its state changes.



Variant: Notify may be private and automatically invoked when updating the state.
Variant: Observers may be stateless.

State (Objects for States)

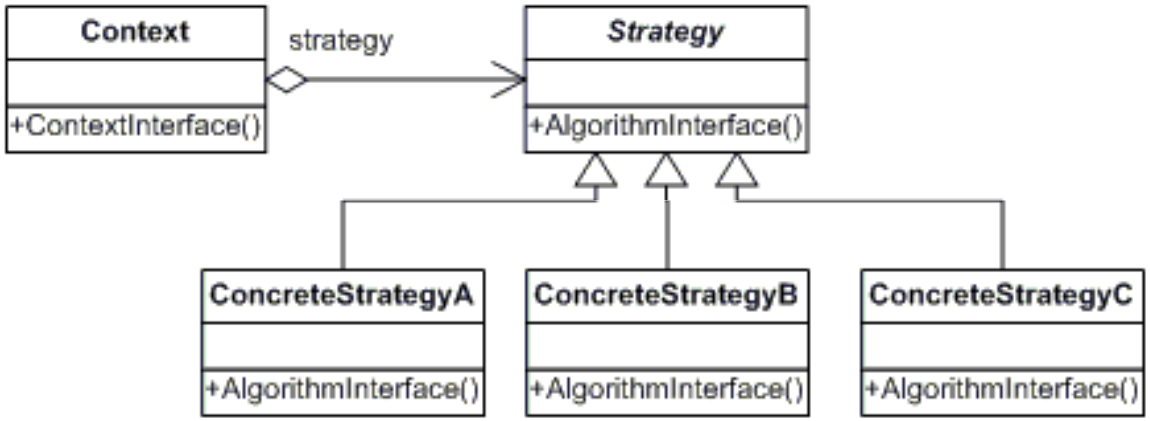
- **Intent**: It allows an object to alter its behavior when its internal state changes. The object will appear to change its class.



- ***Context***: defines the interface of interest to clients and maintains an instance of a ConcreteState subclass that defines the current state.
- ***State***: defines an interface for encapsulating the behavior associated with a particular state of the Context.
- ***ConcreteState***: each subclass implements a behavior associated with a state of the Context.

Strategy (Policy)

- **Intent**: It defines a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

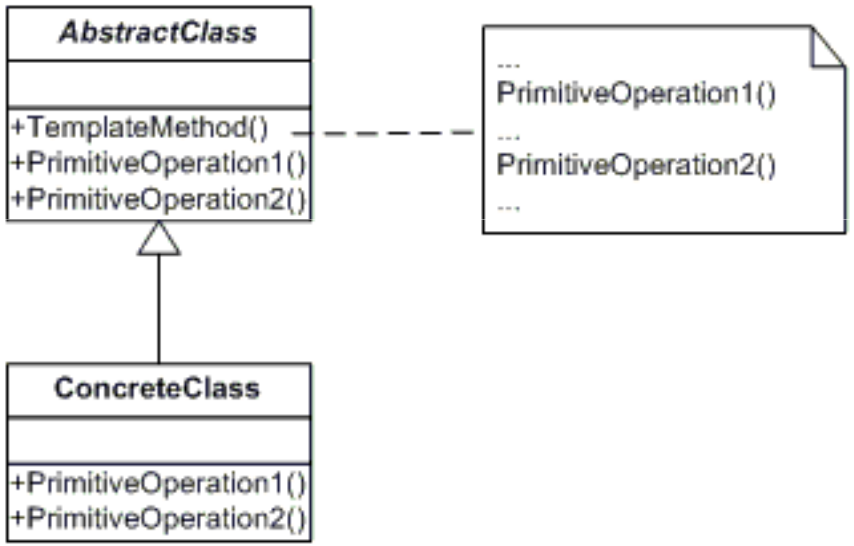


- ***Strategy***: declares an interface common to all supported algorithms. Context uses this interface to call the algorithm defined by a ConcreteStrategy.
- ***ConcreteStrategy***: implements the algorithm using the Strategy interface.
- ***Context***: is configured with a ConcreteStrategy object, maintains a reference to a Strategy object, and may define an interface that lets Strategy access its data.

Template Method

- **Intent**: It defines the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

- ***AbstractClass***: defines abstract primitive operations that concrete subclasses define to implement steps of an algorithm, implements a template method defining the skeleton of an algorithm. The template method calls primitive operations as well as operations defined in AbstractClass or those of other objects
- ***ConcreteClass***: implements the primitive operations to carry out subclass-specific steps of the algorithm



Variant: Hook operations can be plugged into the algorithm, so that a client can specialize the algorithm to perform some additional steps in some particular points of the base algorithms.

Visitor

- **Intent**: It represents an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

- ***Visitor***: declares a Visit operation for each class of ConcreteElement in the object structure. The operation's name and signature identifies the class that sends the Visit request to the visitor. That lets the visitor determine the concrete class of the element being visited. Then the visitor can access the element directly through its particular interface.
- ***ConcreteVisitor***: implements each operation declared by Visitor. Each operation implements a fragment of the algorithm defined for the corresponding class of object in the structure. ConcreteVisitor provides the context for the algorithm and stores its local state. This state often accumulates results during the traversal of the structure.
- ***Elements***: defines an Accept operation that takes a visitor as an argument.
- ***ConcreteElements***: implements an Accept operation that takes a visitor as an argument
- ***ObjectStructure***: can enumerate its elements, may provide a high-level interface to allow the visitor to visit its elements, and may either be a Composite or a collection such as a list or a set