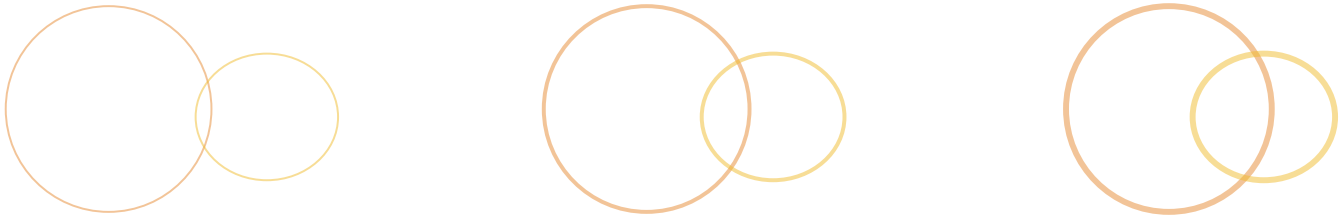


Behavioral Patterns

What are Behavioral Patterns

- Describe algorithms, assignment of responsibility, and interactions between objects (behavioral relationships)
  - Behavioral class patterns use inheritence to distribute behavior
  - Behavioral object patterns use composition
- General example:
  - Model-view-controller in UI application
  - Iterating over a collection of objects
  - Comparable interface in Java

List of Structural Patterns

- Class scope pattern:
  - Interpreter
  - Template Method
- Object scope patterns:
  - Chain of Responsibility
  - Command
  - Iterator
  - Mediator
  - Memento
  - Observer
  - State
  - Strategy
  - Visitor

CoR Pattern Description

- Intent: Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.
- AKA: Handle/Body
- Motivation: User Interfaces function as a result of user interactions, known as events. Events can be handled by a component, a container, or the operating system. In the end, the event handling should be decoupled from the component.
- Applicability:
  - more than one object may handle a request, and the handler isn't known *a priori.*
  - Want to issue a request to one of several objects without specifying the receiver

CoR Real World Example

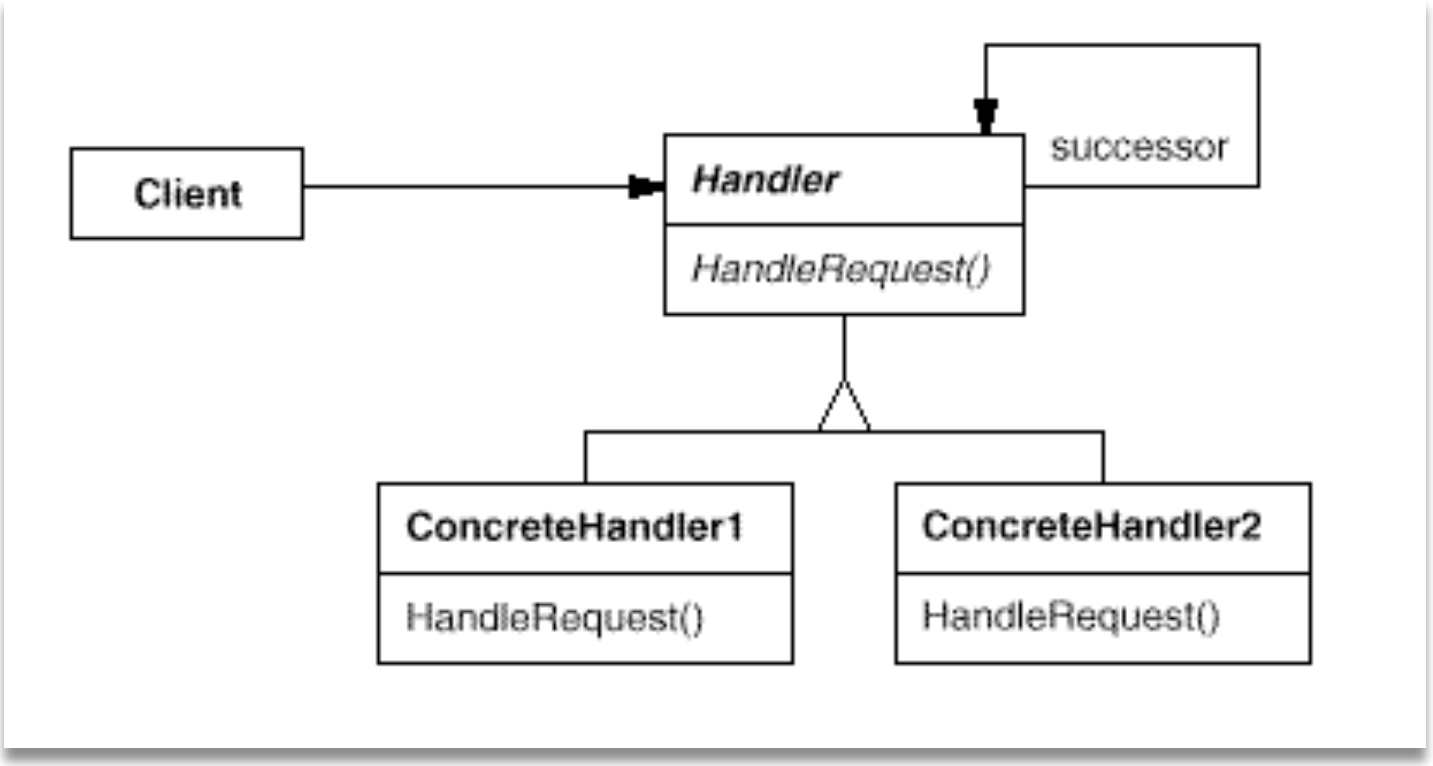
- The *Chain of Responsibility pattern avoids coupling the sender of a request to the receiver, by giving more than one object a chance to handle the request.*

- *Mechanical coin sorting banks use the Chain of Responsibility. Rather than having a separate slot for each coin denomination coupled with receptacle for the denomination, a single slot is used. When the coin is dropped, the coin is routed to the appropriate receptacle by the mechanical mechanisms within the bank.*

CoR Software Example

- Event Handling mechanism pre Java 1.1

CoR Pattern UML



Command Pattern Description

- Intent: Encapsulate a request as an object, letting you parameterize clients with different requests
- AKA: Action / Transaction
- Motivation: You need to be able to change the implementation of an operation dynamically, at run-time by specifying an operation as an object
- Applicability:
  - Decouple implementation of operation
  - Creation consistent action structure

Command Real World Example

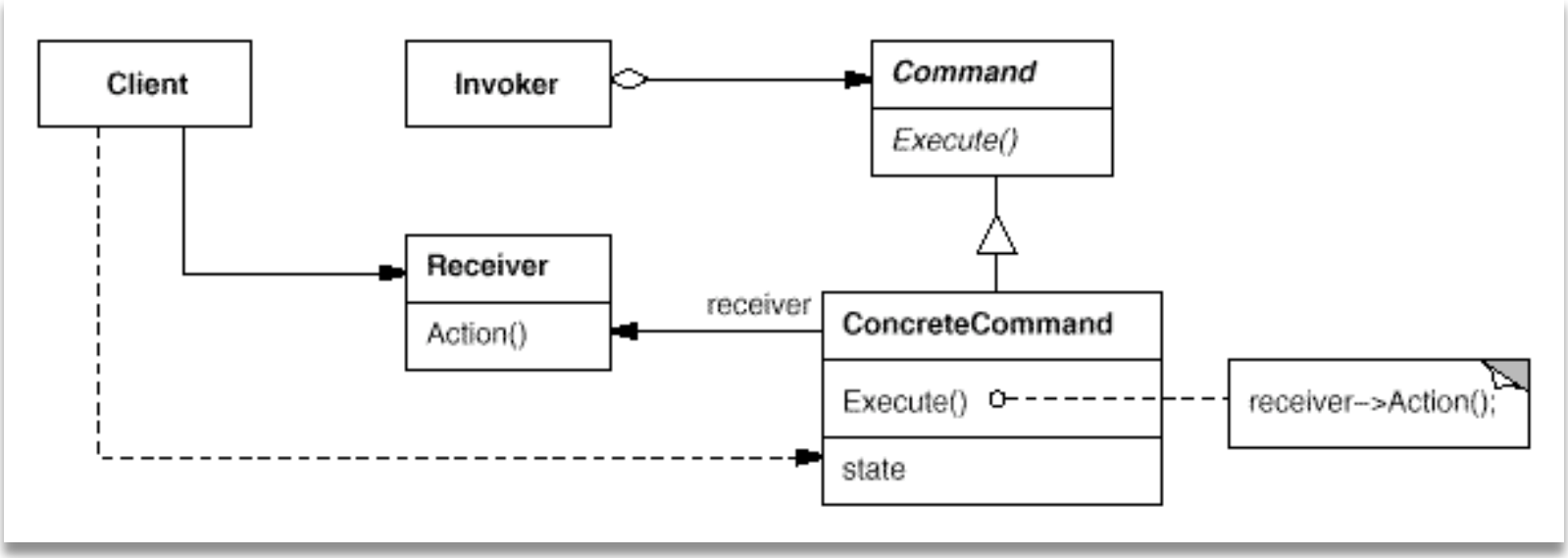
- The *Command pattern allows requests to be encapsulated as objects, thereby allowing clients to be paramaterized with different requests.*

- *The "check" at a diner is an example of a Command pattern. The waiter or waitress takes an order, or command from a customer, and encapsulates that order by writing it on the check. The order is then queued for a short order cook. Note that the pad of "checks" used by different diners is not dependent on the menu, and therefore they can support commands to cook many different items.*

Command Software Example

- Java's Threading System
- Java's Swing Action commands
- Struts Action commands

Command Pattern UML



Interpreter Pattern Description

- Intent: Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.
- AKA: N/A
- Motivation: If a particular kind of problem occurs often enough, then it might be worthwhile to express instances of the problem as sentences in a simple language. Then you can build an interpreter that solves the problem by interpreting these sentences.
- Applicability:
  - You need to interpret a language and language can be represented as syntax trees.

Interpreter Real World Example

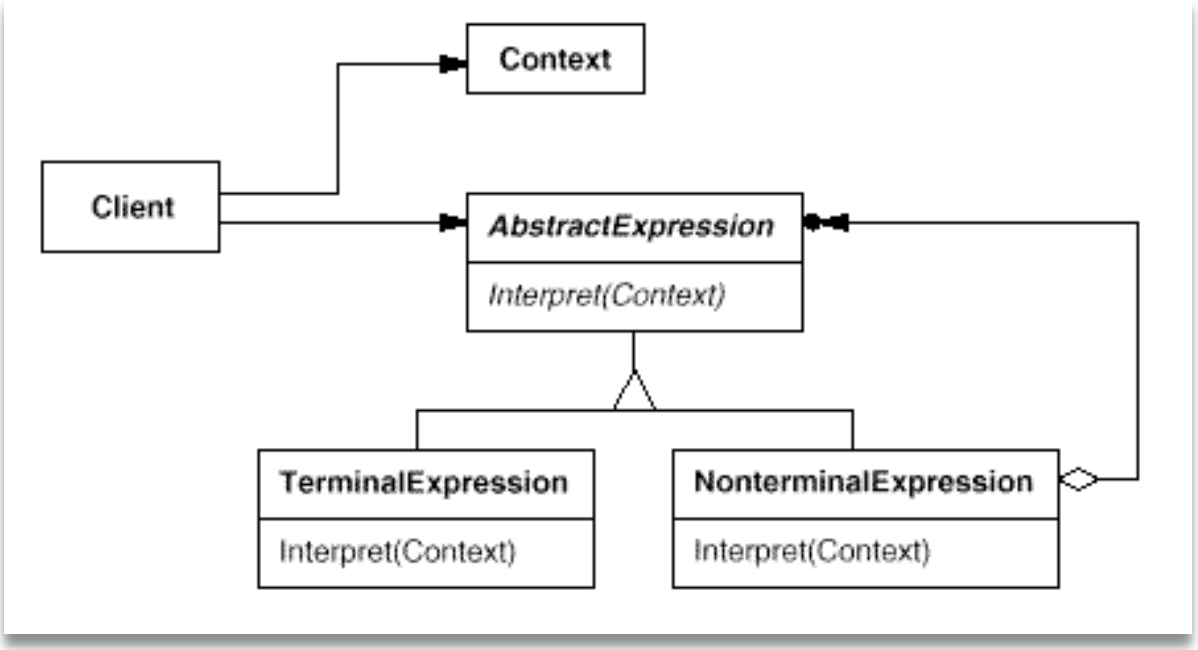
- The *Interpreter pattern defines a grammatical representation for a language and an interpreter to interpret the grammar.*

- *Musicians are examples of Interpreters. The pitch of a sound and its duration can be represented in musical notation on a staff. This notation provides the language of music. Musicians playing the music from the score are able to reproduce the original pitch and duration of each sound represented.*

Interpreter Software Example

- HTML and XML Parsers

Interpreter Pattern UML



Iterator Pattern Description

- Intent: Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
- AKA: Cursor
- Motivation: A list should give you a way to access its elements without exposing its internal structure. You want to traverse the list in different ways, but you don't want to have all the traversal operations defined in the list interface.
- Applicability:
  - Access an aggregate object's contents without exposing its internal representation
  - Support multiple traversals of aggregate objects

Iterator Real World Example

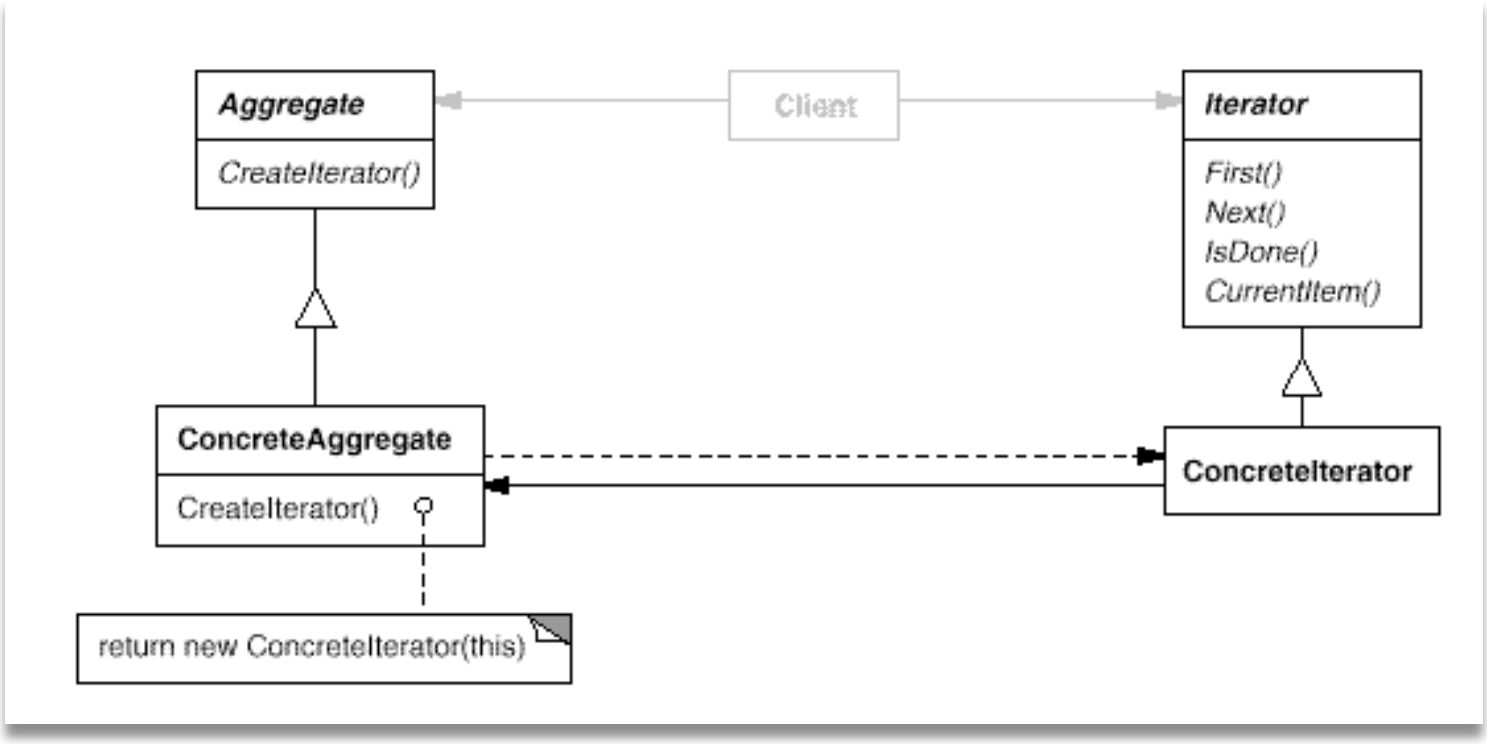
- The *Iterator provides ways to access elements of an aggregate object sequentially without exposing the underlying structure of the object.*

- *On early television sets, a dial was used to change channels. When channel surfing, the viewer was required to move the dial through each channel position, regardless of whether or not that channel had reception. On modern television sets, a next and previous button are used. When the viewer selects the "next" button, the next tuned channel will be displayed. Consider watching television in a hotel room in a strange city. When surfing through channels, the channel number is not important, but the programming is. If the programming on one channel is not of interest, the viewer can request the next channel, without knowing its number.*

Iterator Software Example

- Java Collections API

Iterator Pattern UML



Mediator Pattern Description

- Intent: Define 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.
- AKA: N/A
- Motivation: There are a large number of classes and objects that collaborate with one another. Managing those collaborations is too complex.
- Applicability:
  - Set of objects communicate in well-defined but complex ways. The resulting interdependencies are unstructured and difficult to understand.
  - Reusing an object is difficult because it refers to and communicates with many other objects.

Mediator Real World Example

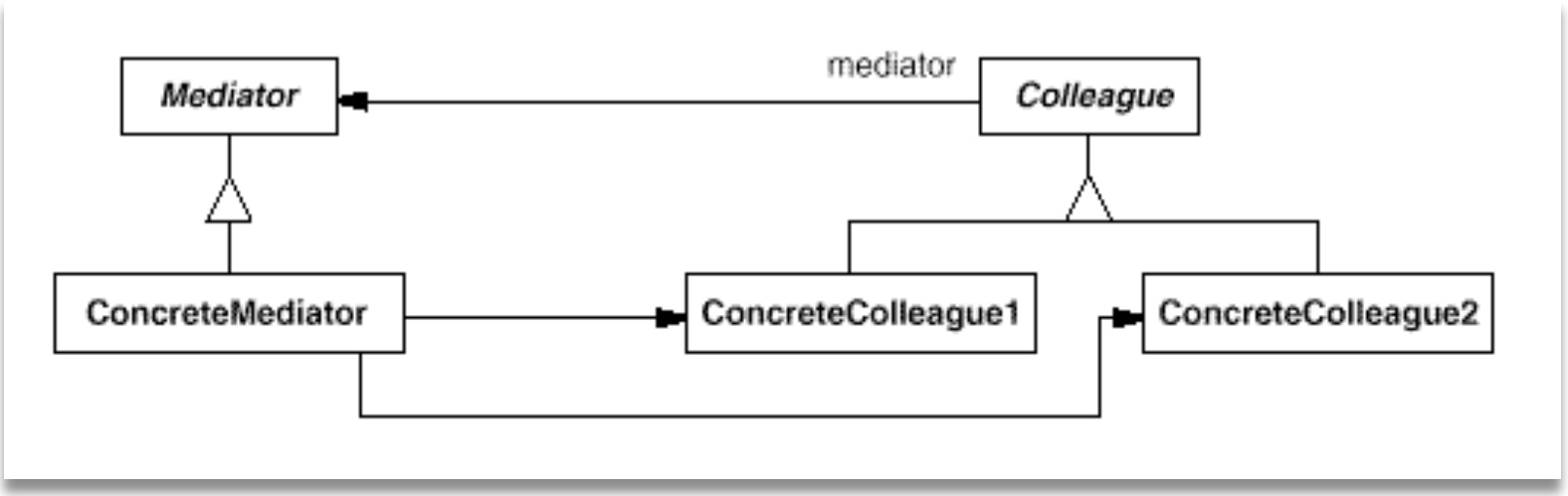
- The *Mediator defines an object that controls how a set of objects interact. Loose coupling between colleague objects is achieved by having colleagues communicate with the Mediator, rather than with each other.*

- *The control tower at a controlled airport demonstrates this pattern very well. The pilots of the planes approaching or departing the terminal area communicate with the tower, rather than explicitly communicating with one another. The constraints on who can take off or land are enforced by the tower. It is important to note that the tower does not control the whole flight. It exists only to enforce constraints in the terminal area.*

Mediator Software Example

- A bus

Mediator Pattern UML



Memento Pattern Description

- Intent: Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.
- AKA: Token
- Motivation: A web-enabled shopping cart needs to keep track of items you are purchasing so you can fulfill your checkout at a later point in time.
- Applicability:
  - A snapshot of (some portion of) an object's state must be saved so that it can be restored to that state later *and*
  - A direct interface to obtaining the state would expose implementation details and break the object's encapsulation.

Memento Real World Example

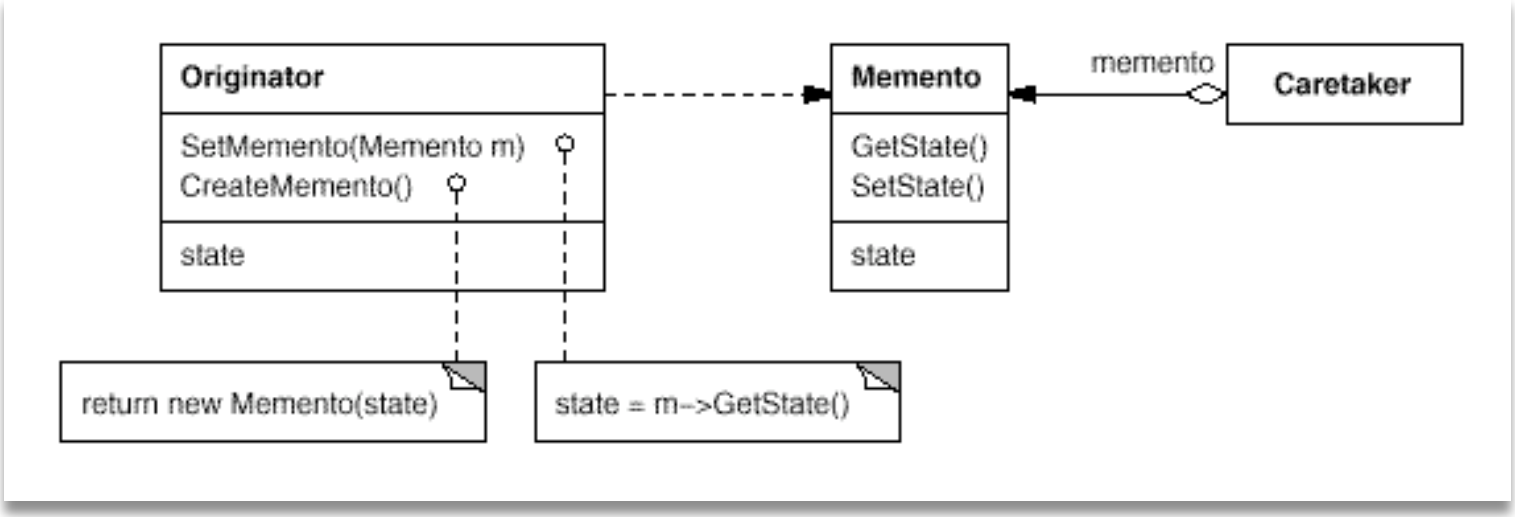
- The *Memento captures and externalizes an object's internal state, so the object can be restored to that state later.*

- *This pattern is common among do-it-yourself mechanics repairing drum brakes on their cars. The drums are removed from both sides, exposing both the right and left brakes. Only one side is disassembled, and the other side serves as a Memento of how the brake parts fit together. Only after the job has been completed on one side is the other side disassembled. When the second side is disassembled, the first side acts as the Memento.*

Memento Software Example

- HTTP Sessions
- Cookies
- Serialized Objects

Memento Pattern UML



Observer Pattern Description

- Intent: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically
- AKA: Dependents, Publish-Subscribe
- Motivation: An object-relational-mapping framework needs to keep track of changes made in a database and map them to an object and vice versa
- Applicability:
  - When a change to one object requires changing others, and you don't know how many objects need to be changed.
  - When an object should be able to notify other objects without making assumptions about who these objects are

Observer Real World Example

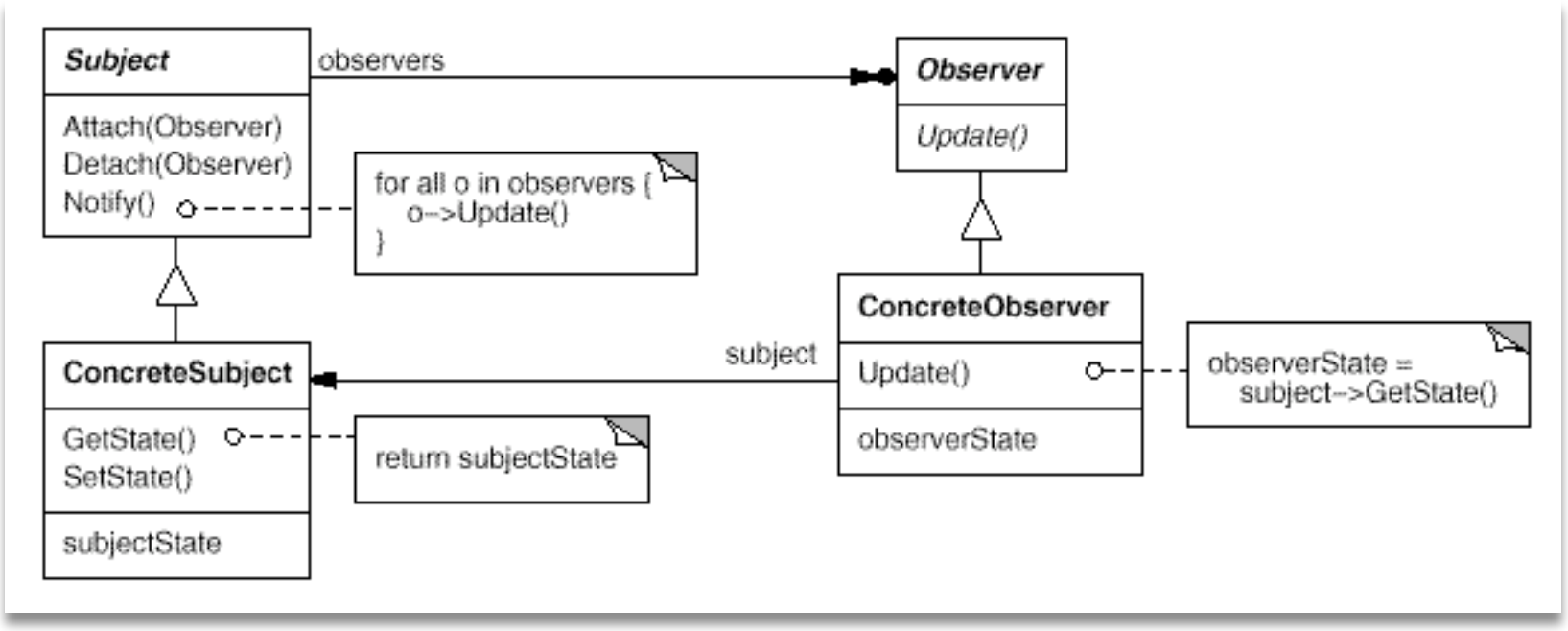
- The *Observer defines a one to many relationship, so that when one object changes state, the others are notified and updated automatically.*

- *Some auctions demonstrate this pattern. Each bidder possesses a numbered paddle that is used to indicate a bid. The auctioneer starts the bidding, and "observes" when a paddle is raised to accept the bid. The acceptance of the bid changes the bid price, which is broadcast to all of the bidders in the form of a new bid.*

Observer Software Example

- Most user interface toolkits
- Object-relational-mapping framworks

Observer Pattern UML



State Pattern Description

- Intent: Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.
- AKA: N/A
- Motivation: Consider a class TCPConnection that represents a network connection. A TCPConnection object can be in one of several different states: Established, Listening, Closed. When a TCPConnection object receives requests from other objects, it responds differently depending on its current state.
- Applicability:
  - An object's behavior depends on its state, and it must change its behavior at run-time depending on that state.

State Real World Example

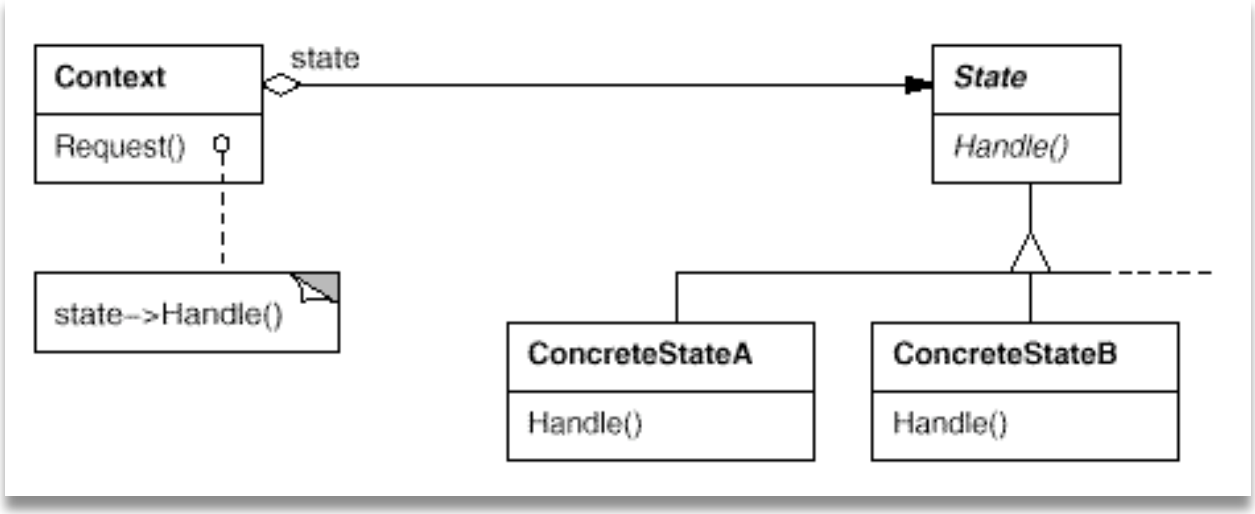
- The *State pattern allows an object to change its behavior when its internal state changes.*

- *This pattern can be observed in a vending machine. Vending machines have states based on the inventory, amount of currency deposited, the ability to make change, the item selected, etc. When currency is deposited and a selection is made, a vending machine will either deliver a product and no change, deliver a product and change, deliver no product due to insufficient currency on deposit, or deliver no product due to inventory depletion.*

State Software Example

- Web Servers
- Real-time software

State Pattern UML



Strategy Pattern Description

- Intent: Define a family of algorithms, encapsulate each one, and make them interchangeable.
- AKA: Policy
- Motivation: A system needs to change how it connects to the web depending on the underlying security within an organization. It may be able to connect directly, use a proxy, or require security.
- Applicability:
  - You need different variants of an algorithm.

Strategy Real World Example

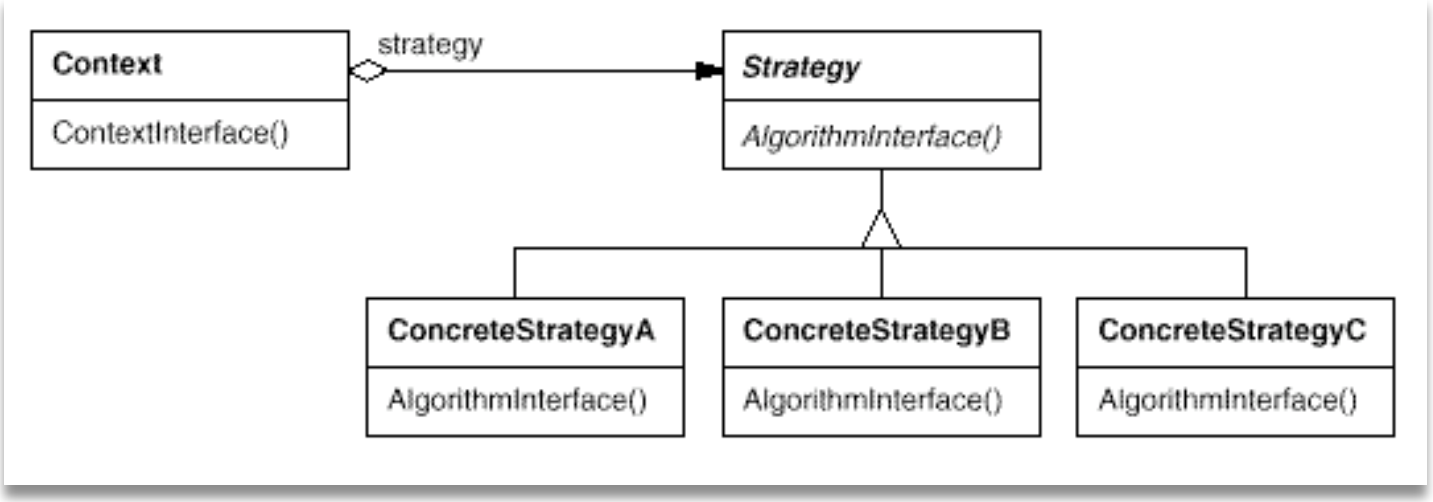
- A *Strategy defines a set of algorithms that can be used interchangeably.*

- *Modes of transportation to an airport is an example of a Strategy. Several options exist, such as driving one's own car, taking a taxi, an airport shuttle, a city bus, or a limousine service. For some airports, subways and helicopters are also available as a mode of transportation to the airport. Any of these modes of transportation will get a traveler to the airport, and they can be used interchangeably. The traveler must chose the Strategy based on tradeoffs between cost, convenience, and time.*

Strategy Software Example

- Comparator / Comparable interfaces in Java collections

Strategy Pattern UML



Template Method Pattern Description

- Intent: Define the skeleton of an algorithm in an operation, deferring some steps to subclasses
- AKA: N/A
- Motivation: A data model form an application has no inherent knowledge of how its used. The algorithm for working with this data is delegated to a template. Different templates could work with the same data in different ways.
- Applicability:
  - Implement the invariant parts of an algorithm once and leave it up to subclasses to implement the behavior that can vary

Template Method Real World Example

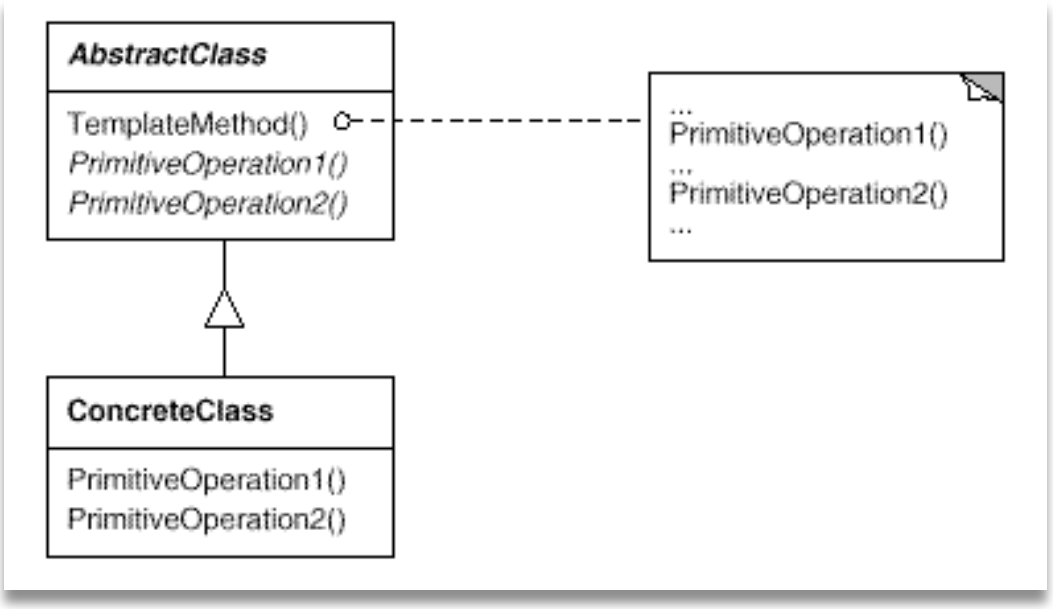
- The Template Method defines a skeleton of an algorithm in an operation, and defers some steps to subclasses.

- Home builders use the Template Method when developing a new subdivision. A typical subdivision consists of a limited number of floor plans, with different variations available for each floor plan. Within a floor plan, the foundation, framing, plumbing, and wiring will be identical for each house. Variation is introduced in the latter stages of construction to produce a wider variety of models.

Template Method Software Example

- Template-oriented websites
- Content management systems

Template Method Pattern UML



Visitor Pattern Description

- Intent: Represent 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
- AKA: Agent
- Motivation: User interfaces are made up of a composition of components. Each component may contain other components, which may contain other components.
- Applicability:
  - You want to represent part-whole hierarchies of objects
  - You want clients to be able to ignore the difference between compositions of objects and individual objects. Clients will treat all objects in the composite structure uniformly.

Visitor Real World Example

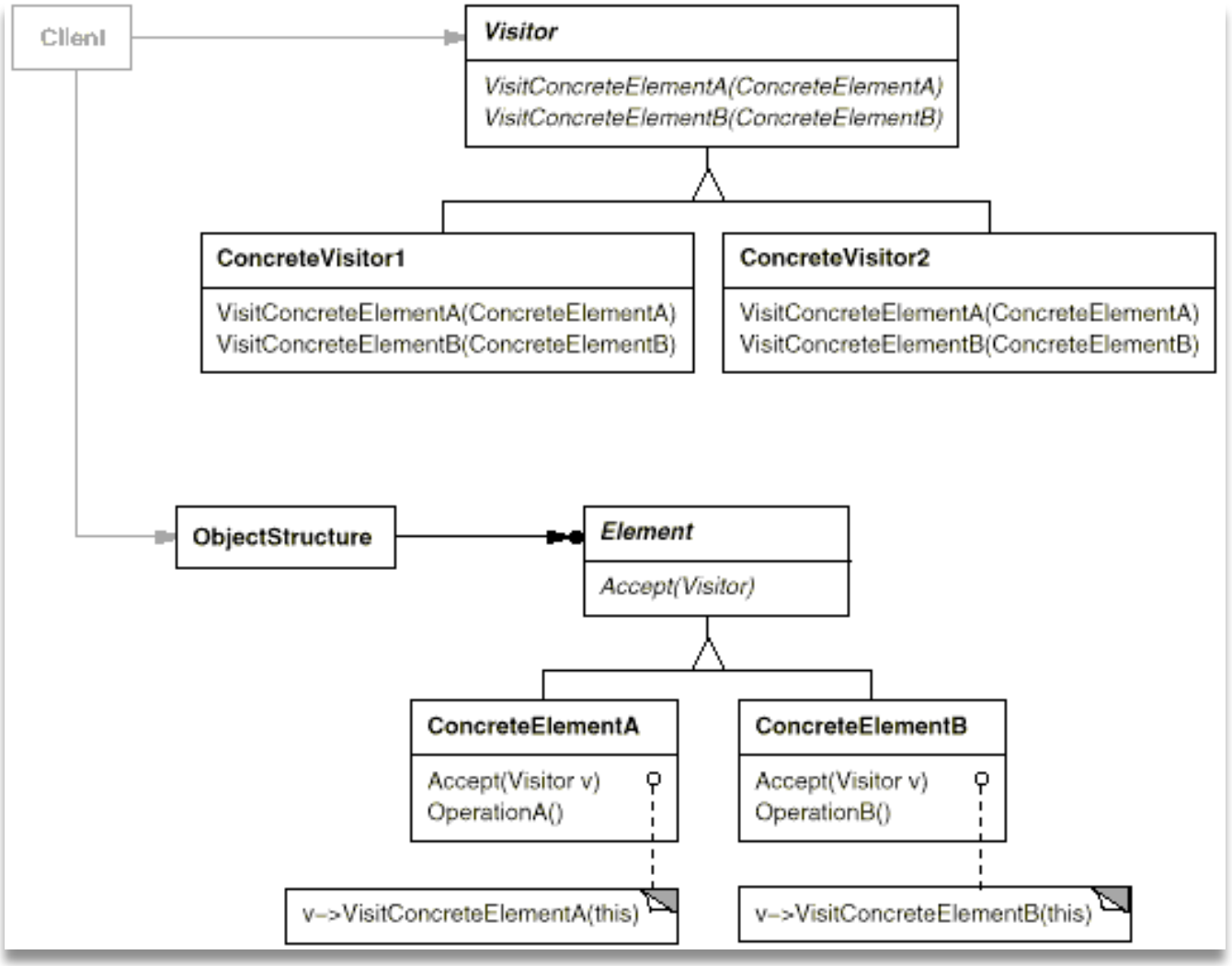
- The *Visitor pattern represents an operation to be performed on the elements of an object structure, without changing the classes on which it operates.*

- *This pattern can be observed in the operation of a taxi company. When a person calls a taxi company he or she becomes part of the company's list of customers. The company then dispatches a cab to the customer (accepting a visitor). Upon entering the taxi, or Visitor, the customer is no longer in control of his or her own transportation, the taxi (driver) is.*

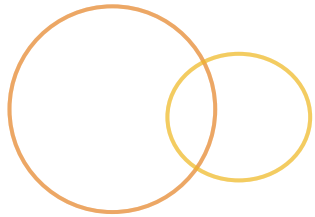
Visitor Software Example

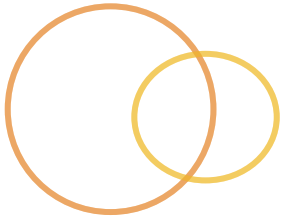
- ?

Visitor Pattern UML



Summary



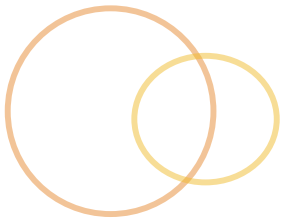


- Interpreter
- Template Method
- Chain of Responsibility
- Command
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Visitor

About DevelopIntelligence

- Founded in 2003
- Provides outsourced services to learning organizations in area of software development
- Represents over 35 years of combined experience, enabling software development community through educational and performance services
- Represents over 50 years of combined software development experience
- Delivered training to over 40,000 developers worldwide

Areas of Expertise



- Instruction
  - Java
  - J2EE
  - WebServices / SOA
  - Web Application Development
  - Database Development
  - Open Source Frameworks
  - Application Servers
- Courseware
  - Java Application Development
  - Java Web App Development
  - Enterprise Java Development
  - OOAD / UML
  - IT Managerial
  - Emerging Technologies and Frameworks

Contact Us

- For more information about our services, please contact us:
  - Kelby Zorgdrager
  - Kelby@DevelopIntelligence.com
  - 303-395-5340