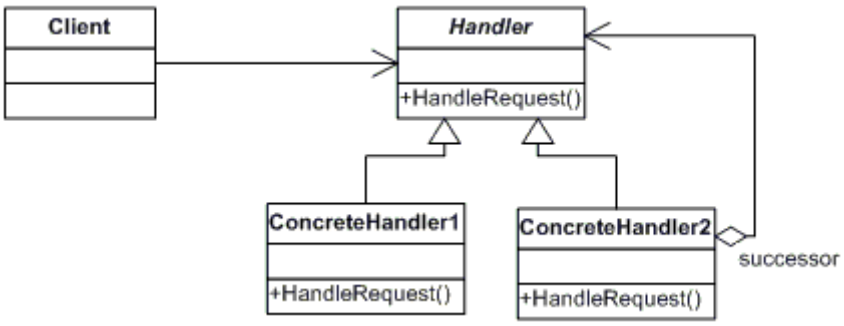
PATRONES DE COMPORTAMIENTO

- **Chain of Responsability** (Frecuencia de uso: Media baja)
  Evita acoplar el remitente de una solicitud a su receptor, dándole a más de un objeto la oportunidad de manejar la solicitud. Encadena los objetos receptores y pasa la solicitud a lo largo de la cadena hasta que un objeto lo maneje.



Código estructural -> varios obj vinculados (cadena) se le ofrece la oportunidad de responder a una solicitud y entregarla al siguiente obj de la línea (sucesor).

Salida

ConcreteHandler1 handled request 2
ConcreteHandler1 handled request 5
ConcreteHandler2 handled request 14
ConcreteHandler3 handled request 22
ConcreteHandler2 handled request 18
ConcreteHandler1 handled request 3
ConcreteHandler3 handled request 27
ConcreteHandler3 handled request 20

- Ejemplo

Varios gerentes o ejecutivos puedan responder a una solicitud de compra o entregarla a un superior. Reglas según puesto.

Participantes

Las clases y objetos que participan en este patrón son:

- Handler (Approver)
  - define una interfaz para manejar las solicitudes
  - (opcional) implementa el enlace sucesor
- ConcreteHandler (Director, VicePresident, President)
  - maneja las solicitudes de las que es responsable
  - puede acceder a su sucesor
  - si ConcreteHandler puede manejar la solicitud, lo hace; De lo contrario, reenvía la solicitud a su sucesor
- Client (ChainApp)
  - inicia la solicitud a un objeto ConcreteHandler en la cadena

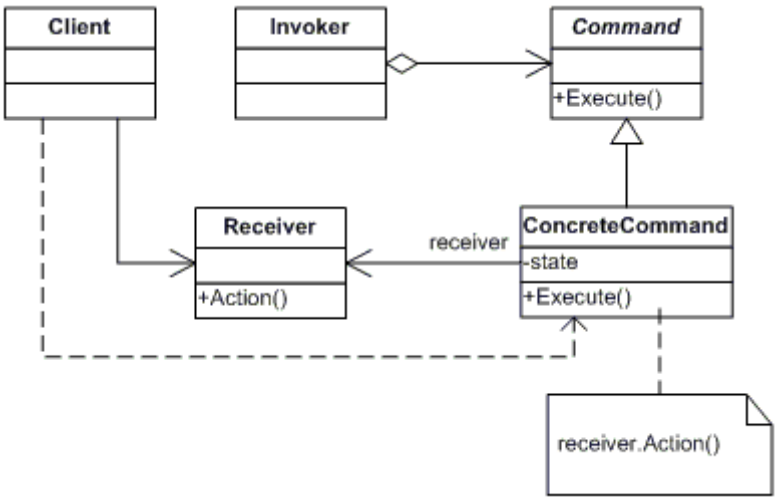
Salida

Director Larry approved request# 2034

President Tammy approved request# 2035

Request# 2036 requires an executive meeting!

- **Command** (Frecuencia de uso: Media alta)
  Encapsula una solicitud como un objeto, lo que le permite parametrizar a los clientes con diferentes solicitudes, solicitudes de cola o de registro y admite operaciones que no se pueden deshacer.



Código estructural -> almacenar solicitudes como objectos permitiendo a los clientes ejecutarlas o reproducirlas. ConcreteCommand: enlace entre obj Receiver y una acción

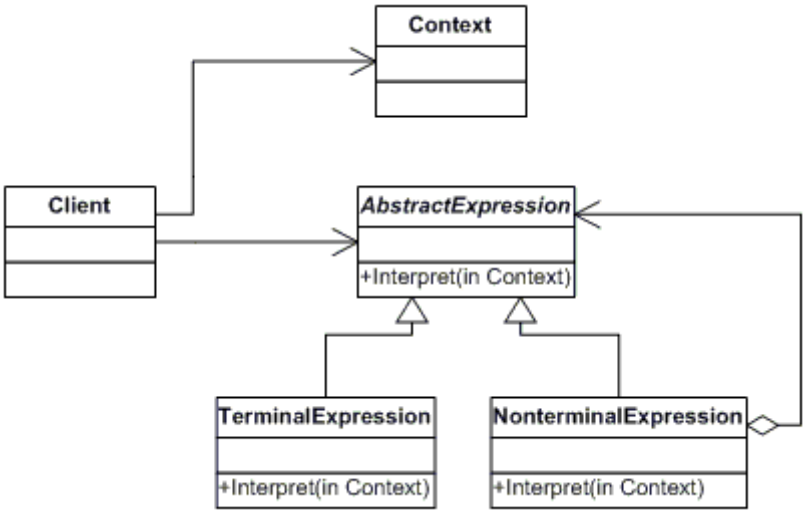
  - Ejemplo

Participantes

Las clases y objetos que participan en este patrón son:

- Command (Command)
  - declara una interfaz para ejecutar una operación
- ConcreteCommand (CalculatorCommand)
  - define un enlace entre un objeto Receiver y una acción
  - implementa Execute invocando la(s) operación(es) correspondiente (s) en el Receiver
- Client (CommandApp)
  - crea un objeto ConcreteCommand y configura su receptor
- Invoker (User)
  - pide al comando que realice la solicitud
- Receiver (Calculator)
  - sabe cómo realizar las operaciones asociadas a la realización de la solicitud.

- **Interpreter** (Frecuencia de uso: Baja)
  Dado un lenguaje, define una representación para su gramática junto con un intérprete que usa la representación para interpretar sentencias en el idioma.



Código estructural

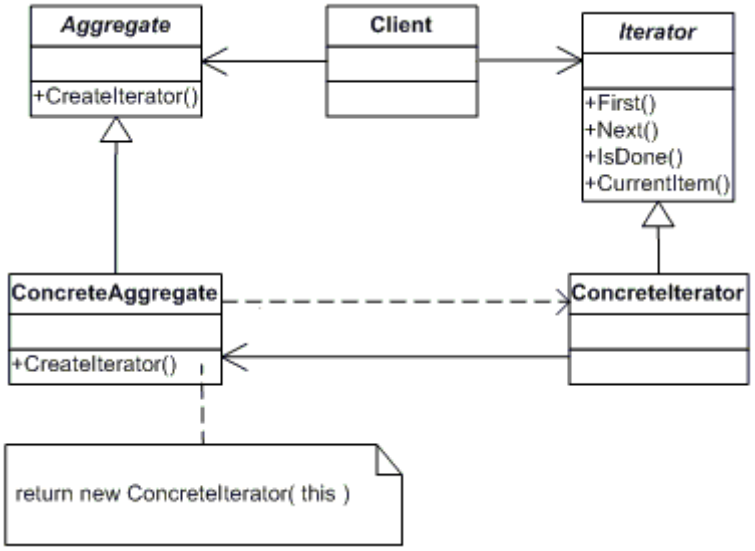
- Ejemplo -> transformar nº romano a decimal

Participantes

Las clases y objetos que participan en este patrón son:

- AbstractExpression (Expression)
  - declara una interfaz para ejecutar una operación
- TerminalExpression (ThousandExpression, HundredExpression, TenExpression, OneExpression)
  - implementa una operación de Interpret asociada con los símbolos de terminal en la gramática.
  - Se requiere una instancia para cada símbolo terminal en la oración.
- NonterminalExpression (no usado en el ejemplo)
  - Se requiere una clase de este tipo para cada regla(s) en la gramática
  - mantiene variables de instancia de tipo AbstractExpression para cada uno de los símbolos.
  - implementa una operación de interpretación para símbolos no terminales en la gramática. Normalmente, Interpret se llama a sí mismo de forma recursiva en las variables que representan.
- Context (Context)
  - contiene información que es global para el intérprete
- Client (InterpreterApp)
  - construye (o se da) un árbol de sintaxis abstracta que representa una oración en particular en el lenguaje que define la gramática. El árbol de sintaxis abstracta se ensambla a partir de instancias de las clases NonterminalExpression y TerminalExpression
  - invoca la operación Interpret

- **Iterator** (Frecuencia de uso: Alta)
  Proporciona una forma de acceder a los elementos de un objeto agregado de forma secuencial sin exponer su representación subyacente.



  Código estructural

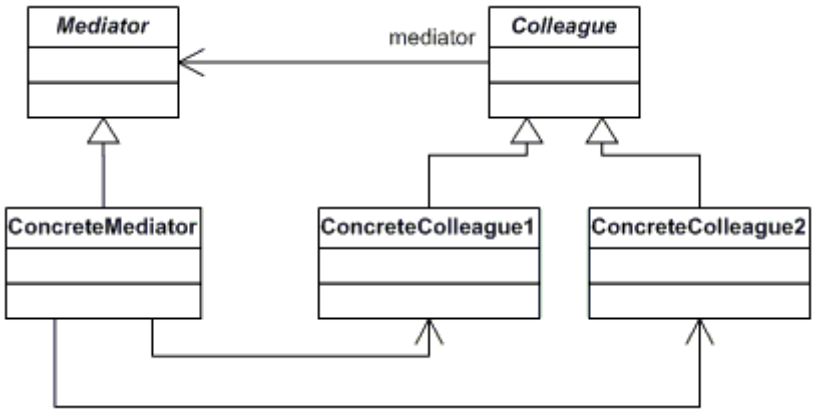
  - Ejemplo -> un tipo bucle for

  Participantes

  Las clases y objetos que participan en este patrón son:

  - Iterator (AbstractIterator)
    - define una interfaz para acceder y recorrer elementos.
  - ConcreteIterator (Iterator)
    - implementa la interfaz Iterator.
    - realiza un seguimiento de la posición actual en el recorrido del agregado.
  - Aggregate (AbstractCollection)
    - define una interfaz para crear un objeto Iterator
  - ConcreteAggregate (Collection)
    - implementa la interfaz de creación de Iterator para devolver una instancia del ConcreteIterator adecuado

- **Mediator** (Frecuencia de uso: Media baja)
  Define un objeto que encapsula cómo interactúa un conjunto de objetos. El **Mediator** promueve el bajo acoplamiento evitando que los objetos se refieran entre sí explícitamente, y le permite variar su interacción de forma independiente.



Código estructural

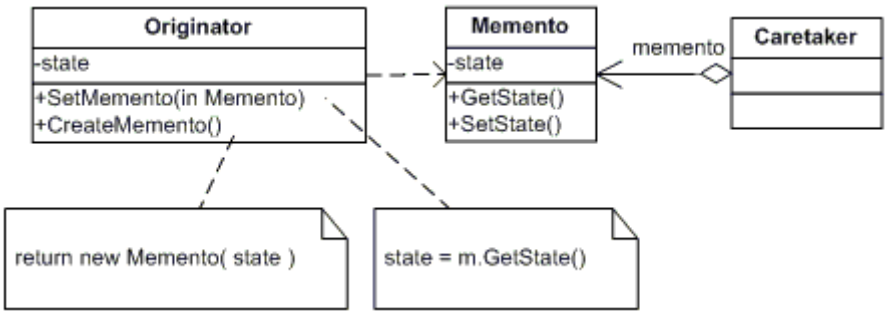
- Ejemplo -> sala de chat

Participantes

Las clases y objetos que participan en este patrón son:

- Mediator (IChatroom) AbstractChatroom
  - define una interfaz para comunicarse con objetos Colleague
- ConcreteMediator (Chatroom)
  - implementa el comportamiento cooperativo mediante la coordinación de los objetos Colleague
  - conoce y mantiene a sus Colleague
- Colleague classes (Participant)
  - Cada clase Colleague conoce su objeto Mediator
  - cada colleague se comunica con su Mediator Cada vez que se hubiera comunicado con otro colleague

- **Memento** (Frecuencia de uso: Baja)
Sin violar la encapsulación, captura y externaliza el estado interno de un objeto para que el objeto pueda restaurarse a este estado más adelante.



Código estructural

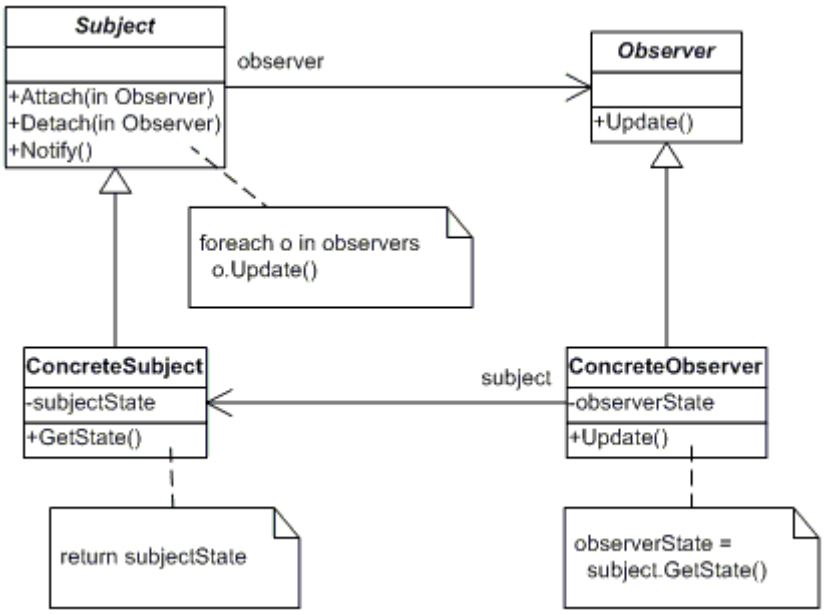
➢ Ejemplo

Participantes

Las clases y objetos que participan en este patrón son:

- Memento (Memento)
  - almacena el estado interno del objeto Originator. El Memento puede almacenar tanto o tan poco del estado interno del originator como sea necesario a discreción del originator.
  - Protege contra el acceso de objetos que no sean el originator. Los mementos tienen efectivamente dos interfaces. El caretaker tiene una interfaz limitada con el Memento: solo puede pasar el memento a los otros objetos. Originator, en cambio, tiene una interfaz amplia, una que le permite acceder a todos los datos necesarios para restablecerse a su estado anterior. Idealmente, solo al Originator que produce el Memento se le permitiría acceder al estado interno del mismo.
- Originator (SalesProspect)
  - crea un Memento que contiene una instantánea de su estado interno actual.
  - usa el Memento para restaurar su estado interno
- Caretaker (ProspectMemory)
  - es responsable de la custodia del Memento
  - nunca opera o examina el contenido de un Memento.

- **Observer** (Frecuencia de uso: Alta)
  Define una dependencia de uno a muchos entre los objetos para que cuando un objeto cambie de estado, todos los objetos que dependan de él sean notificados y actualizados automáticamente.
  Notificar eventos.



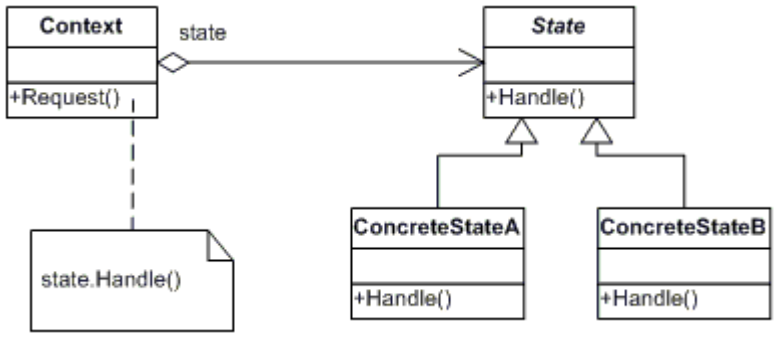
Código estructural

➢ Ejemplo -> se notifica a inversores cada vez que una acción cambia de valor

Participantes

- Subject (Stock)
  - conoce a sus observers. Cualquier número de objetos observer puede observar un subject
  - proporciona una interfaz para adjuntar y separar objetos Observer.
- ConcreteSubject (IBM)
  - almacena el estado de interés para ConcreteObserver
  - envía una notificación a sus Observer cuando cambia su estado
- Observer (IInvestor)
  - define una interfaz de actualización para los objetos que deben ser notificados de los cambios en un subject.
- ConcreteObserver (Investor)
  - mantiene una referencia a un objeto ConcreteSubject
  - almacena el estado que debe ser consistente con las del subject
  - implementa la interfaz de actualización de Observer para mantener su estado coherente con el del subject

- **State** (Frecuencia de uso: Media)
Permite que un objeto altere su comportamiento cuando cambia su estado interno. El objeto aparecerá para cambiar su clase.



Código estructural
Este código estructural demuestra el patrón State que permite que un objeto se comporte de manera diferente dependiendo de su estado interno. La diferencia de comportamiento se delega a objetos que representan este estado.

➢ Ejemplo

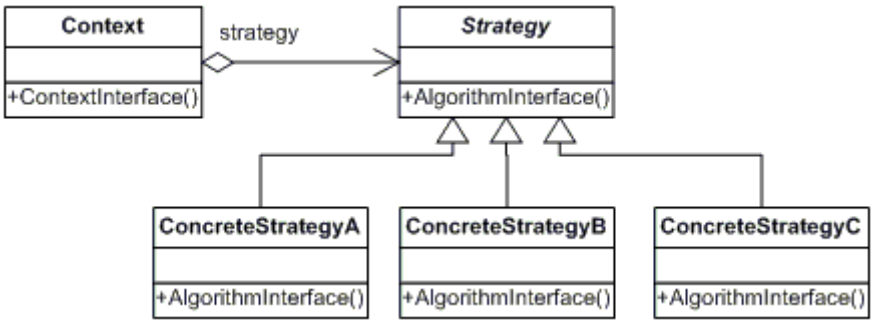
Este código muestra el patrón **State** que permite que una cuenta se comporte de manera diferente dependiendo de su saldo. La diferencia en el comportamiento se delega a los objetos estatales llamados RedState, SilverState y GoldState. Estos estados representan cuentas con descubiertos, cuentas de inicio y cuentas en regla.

Participantes

Las clases y objetos que participan en este patrón son:

- Context (Account)
  - define la interfaz de interés para los clientes
  - mantiene una instancia de una subclase de State que define el estado actual.
- State (State)
  - define una interfaz para encapsular el comportamiento asociado con un estado particular del Context.
- Concrete State (RedState, SilverState, GoldState)
  - cada subclase implementa un comportamiento asociado con un estado de Context

- **Strategy** (Frecuencia de uso: Media alta)
  Define una familia de algoritmos, encapsula cada uno de ellos y los hace intercambiables entre sí. Permite que el algoritmo varíe independientemente de los clientes que lo utilizan.



Código estructural

Este código estructural demuestra el patrón **Strategy** que encapsula la funcionalidad en forma de un objeto

- Ejemplo

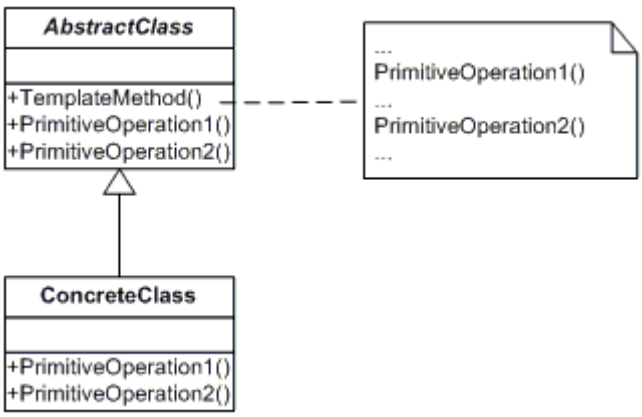
Ordenación de una lista de cadenas.

Participants

Las clases y objetos que participan en este patrón son:

- Strategy (SortStrategy)
  - declara una interfaz común a todos los algoritmos soportados. El contexto usa esta interfaz para llamar al algoritmo definido por ConcreteStrategy
- ConcreteStrategy (QuickSort, ShellSort, MergeSort)
  - implementa el algoritmo usando la interfaz Strategy
- Context (SortedList)
  - se configura con un objeto ConcreteStrategy
  - mantiene una referencia a un objeto Strategy
  - puede definir una interfaz que permita al Strategy acceder a sus datos.

- **Template Method** (Frecuencia de uso: Media)
  Define el esqueleto de un algoritmo en una operación, aplazando algunos pasos a las subclases. Permite a las subclases redefinir ciertos pasos de un algoritmo sin cambiar la estructura del algoritmo.



Código estructural

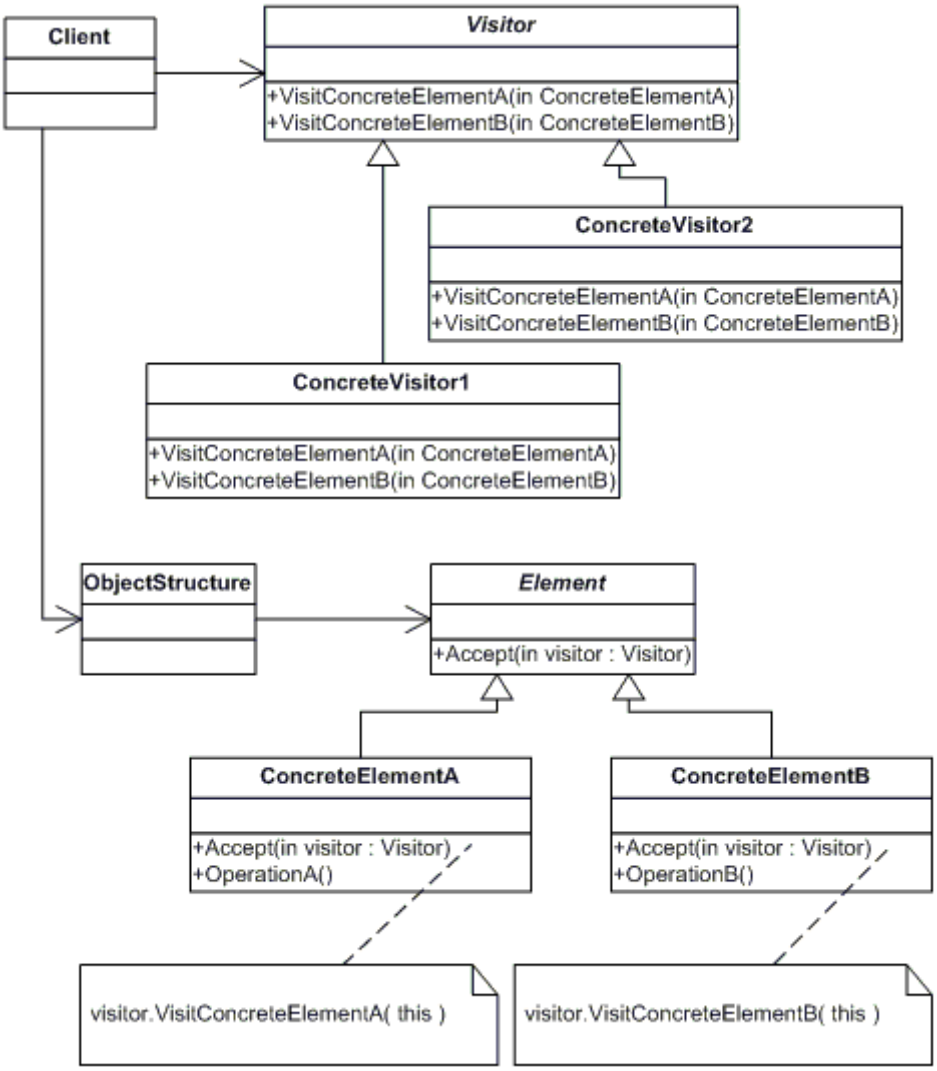
  - Ejemplo

  Participantes

  Las clases y objetos que participan en este patrón son:

  - AbstractClass (DataObject)
    - define operaciones primitivas abstractas que las subclases concretas definen para implementar los pasos de un algoritmo
    - implementa un template method que define el esqueleto de un algoritmo. El template method llama operaciones primitivas así como operaciones definidas en AbstractClass o las de otros objetos.
  - ConcreteClass (CustomerDataObject)
    - implementa las operaciones primitivas para llevar a cabo pasos de subclase específicos del algoritmo

- **Visitor** (Frecuencia de uso: baja)
  Representa una operación a realizar en los elementos de una estructura de objeto. **Visitor** le permite definir una nueva operación sin cambiar las clases de los elementos sobre los que opera.



Código estructural

Este código estructural muestra el patrón **Visitor** en el que un objeto recorre una estructura de objeto y realiza la misma operación en cada nodo de esta estructura.

- Ejemplo

Dos objetos recorren una lista de empleados y realizan la misma operación en cada empleado. Un obj asigna el pago y el otro los días de vacaciones.

Participantes

Las clases y objetos que participan en este patrón son:

- Visitor (IVisitor)
  - declara una operación de Visit para cada clase de ConcreteElement en la estructura del objeto. El nombre y la firma de la operación identifican la clase que envía la solicitud de visit al visitor. Eso permite al visitor determinar la clase concreta del elemento que se está visitando. Entonces el visitor puede acceder a los elementos directamente a través de su interfaz particular

- ConcreteVisitor (IncomeVisitor, VacationVisitor)
  - implementa cada operación declarada por el Visitor. Cada operación implementa un fragmento del algoritmo definido para la clase u objeto correspondiente en la estructura. ConcreteVisitor proporciona el contexto para el algoritmo y almacena su estado local. Este estado a menudo acumula resultados durante el recorrido de la estructura.
- Element (Element)
  - define una operación de aceptación que toma a un Visitor como un argumento.
- ConcreteElement (Employee)
  - implementa una operación que toma a un visitante como un argumento
- ObjectStructure (Employees)
  - puede enumerar sus elementos
  - puede proporcionar una interfaz de alto nivel para permitir al Visitor visitar sus elementos
  - puede ser un Composite (patrón) o una colección como una lista o un conjunto