Patrones GOF

1. Creacionales
   - Abstract Factory
   - Builder
   - Factory Method
   - Prototype
   - Singleton
2. Estructurales
   - Adapter
   - Bridge
   - Composite
   - Decorator
   - Facade
   - Flyweight
   - Proxy
3. De Comportamiento
   - Chain of Responsibility
   - Command
   - Interpreter
   - Iterator
   - Mediator
   - Memento
   - Observer
   - State
   - Strategy
   - Template Method
   - Visitor

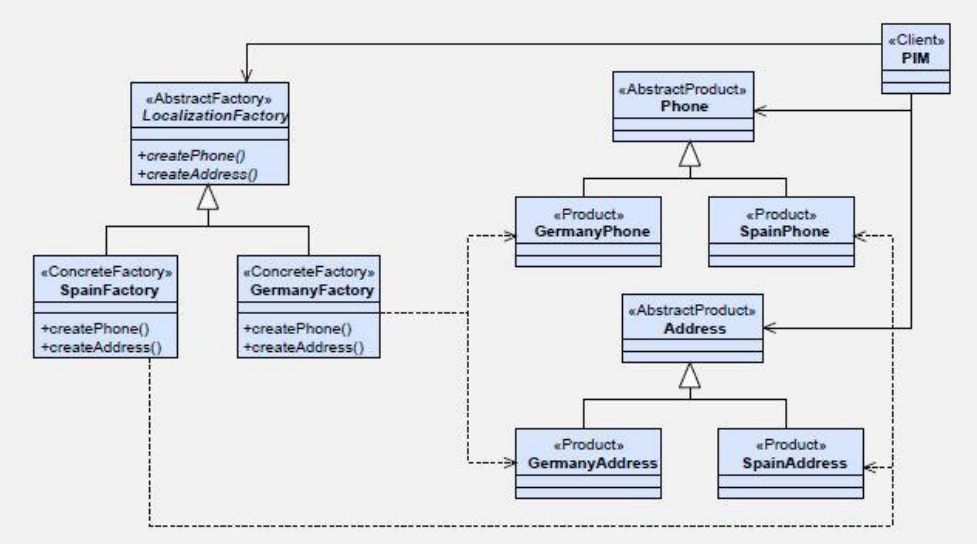
1. Creacionales

- ## Abstract Factory

  Provee una interfaz para crear familias de objetos “producto” relacionados o que dependen entre sí, sin especificar sus clases concretas.

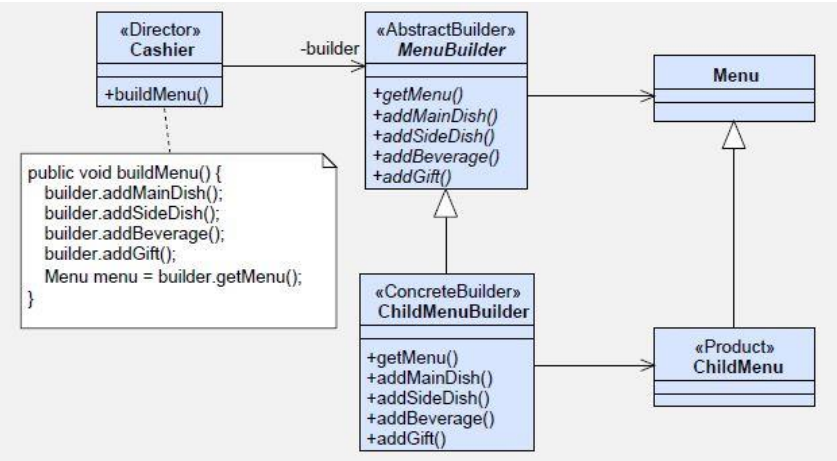
  Uso:

  - Cuando un sistema debe configurarse con una de las múltiples familias de productos
  - Cuando un sistema debe ser independiente de cómo se crean, componen y representan sus productos
  - Cuando los productos de la misma familia deben usarse en conjunto, los productos de familias diferentes no deben usarse juntos y esta restricción debe garantizarse
  - Solo se revelan las interfaces del producto, las implementaciones permanecen ocultas para los clientes



- ## Builder

  Define una instancia para crear un objeto, pero deja que las subclases decidan qué instanciar y permite un control preciso sobre el proceso de construcción.

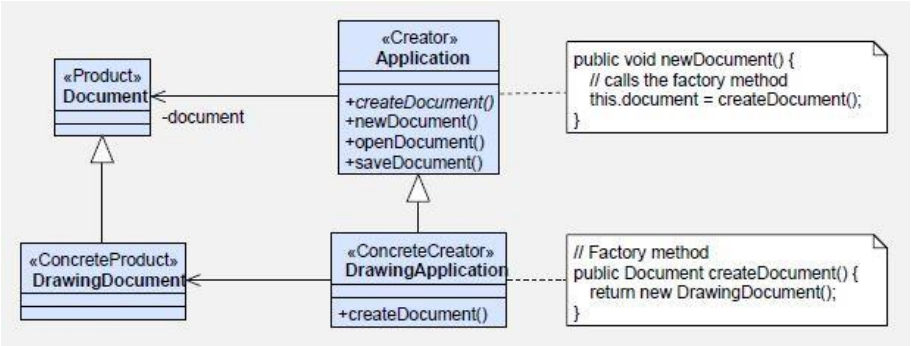


- ## Factory Method

  Define una interfaz para crear objetos, pero deja a las subclases decidir qué clase instanciar y hace referencia al nuevo objeto creado a través de una interfaz común.

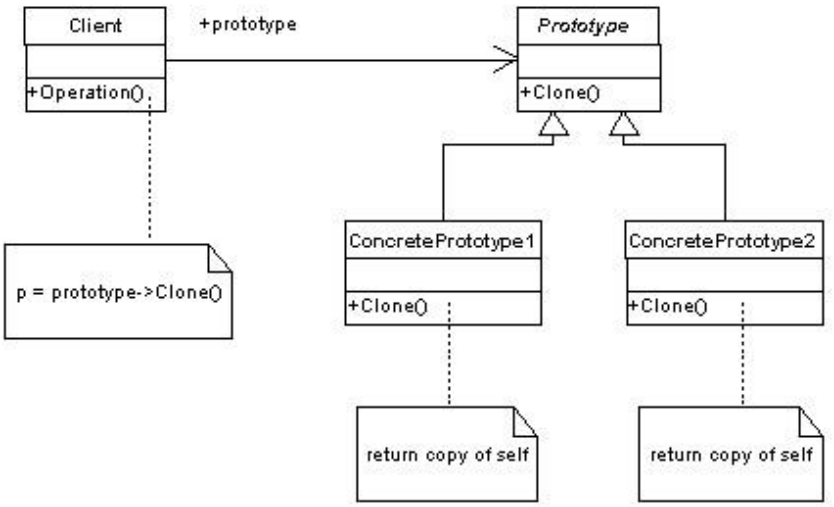
  Uso:

  - Cuando un framework delega la creación de objetos derivados de una superclase común a la fábrica
  - Cuando necesitamos flexibilidad para agregar nuevos tipos de objetos que la clase debe crear
  - Cuando la clase de fábrica base no sabe de qué clases concretas se necesita crear instancias. Delega en sus subclases la creación de objetos concretos
  - Cuando subclases de subclases de fábrica son necesarias para conocer las clases concretas que se deben instanciar



- ## Prototype

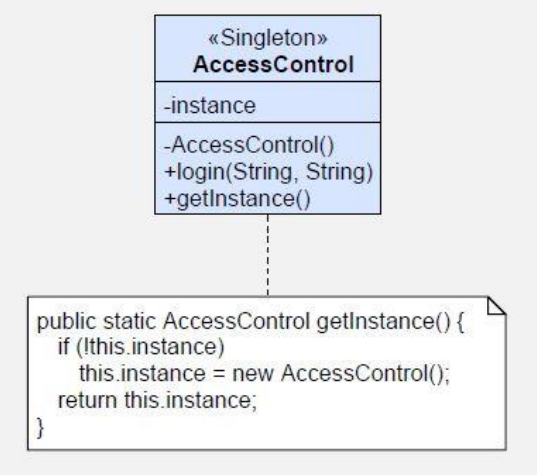
  Especifica los tipos de objetos a crear usando una instancia prototípica y crea nuevos objetos copiando este prototipo.



- Singleton

  Asegura que solo se crea una instancia de la clase y provee un punto de acceso global para el objeto.

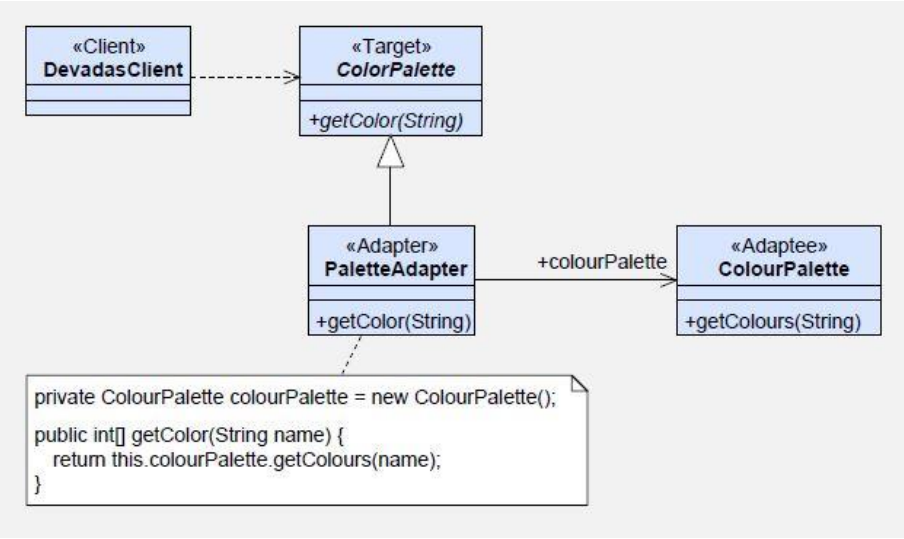
  Uso: Debe usarse cuando debemos asegurarnos de que solo se crea una instancia de una clase y cuando la instancia debe estar disponible a través de todo el código. Se debe tener especial cuidado en entornos milti-threading cuando varios subprocesos deben acceder a los mismos recursos a través del mismo objeto singleton.



2. Estructurales

- Adapter

  Convierte la interfaz de una clase en otra interfaz que los clientes esperan. Adapter permite que clases funcionen juntas, que de otra forma no podría ser debido a interfaces incompatibles.

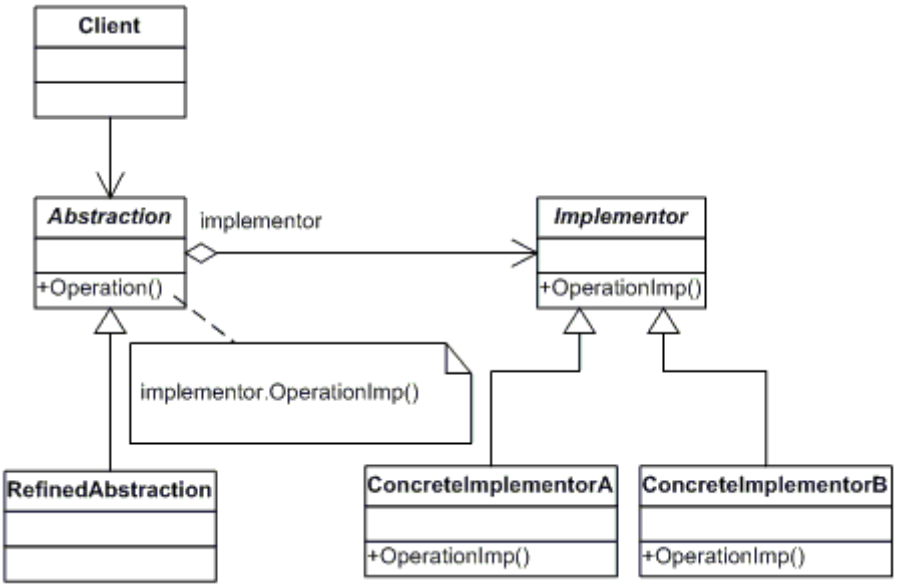


- Bridge

  Desacoplar una abstracción de su implementación, de manera que ambas puedan ser modificadas independientemente. El patrón Bridge permite combinar abstracciones e implementaciones diferentes y extenderlas independientemente.

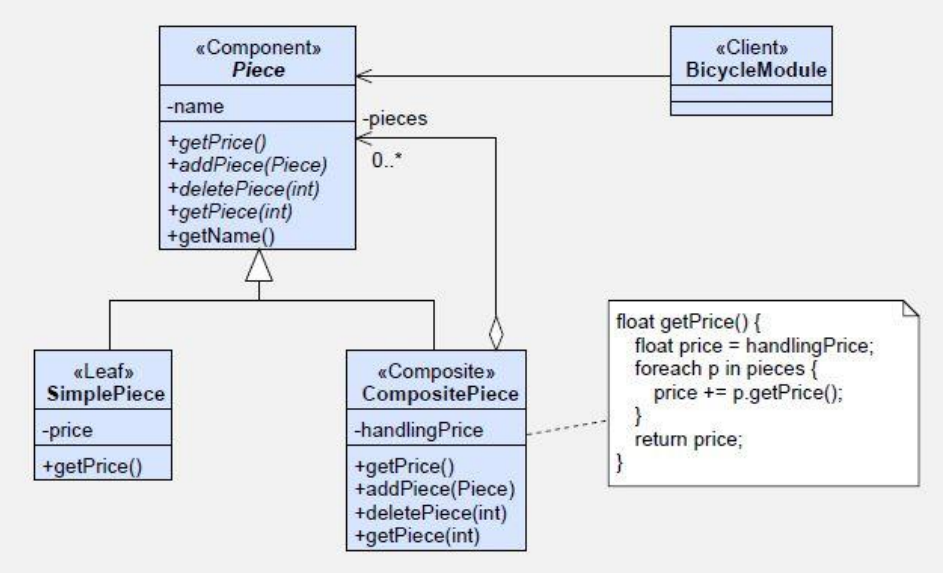
  Uso:

  - Se desea evitar un enlace permanente entre la abstracción y su implementación.
  - Cuando la implementación debe ser seleccionada o cambiada en tiempo de ejecución.
  - Cuando tanto las abstracciones como sus implementaciones deben ser extensibles por medio de subclases.
  - Se desea esconder la implementación de una abstracción completamente a los clientes.
  - Se desea compartir una implementación entre múltiples objetos.



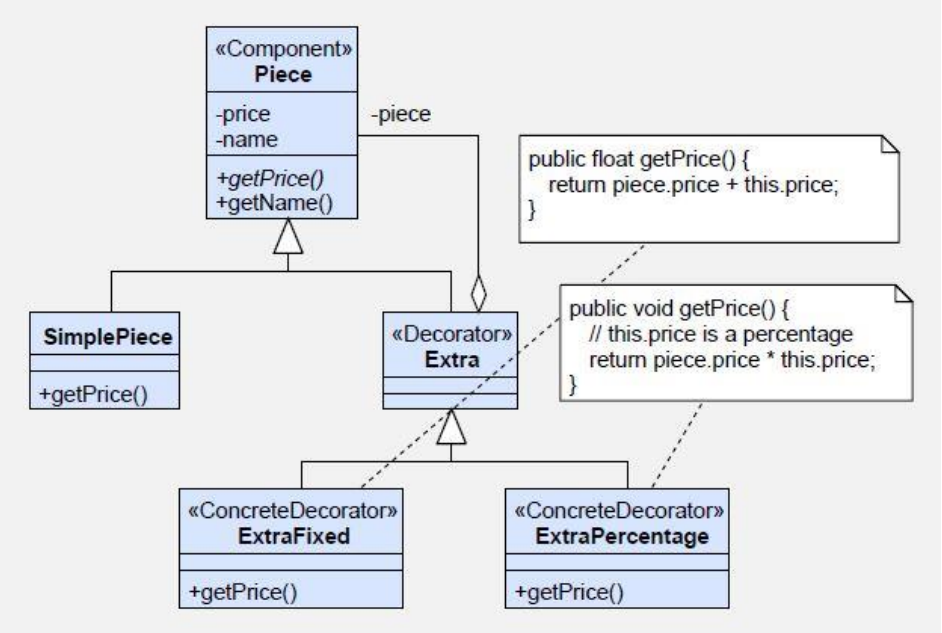
- ## Composite

  Compone objetos en estructuras de árboles para representar jerarquías parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los complejos. Si todas las piezas simples tienen el mismo comportamiento debería evitarse tener una clase para cada pieza simple, sino que es mejor agruparlas en la misma clase.



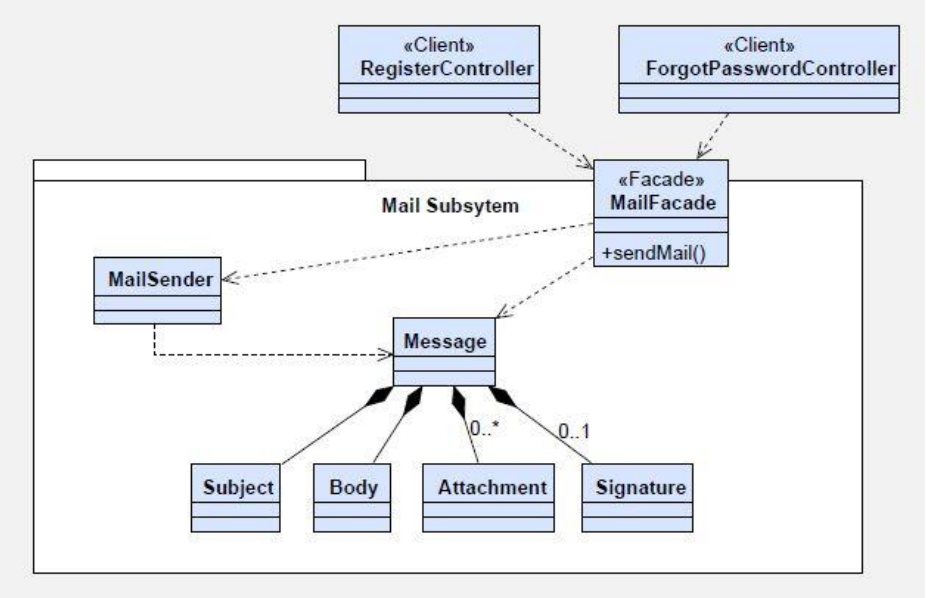
- ## Decorator

  Agregar responsabilidades/funcionalidades adicionales dinámicamente a un objeto.



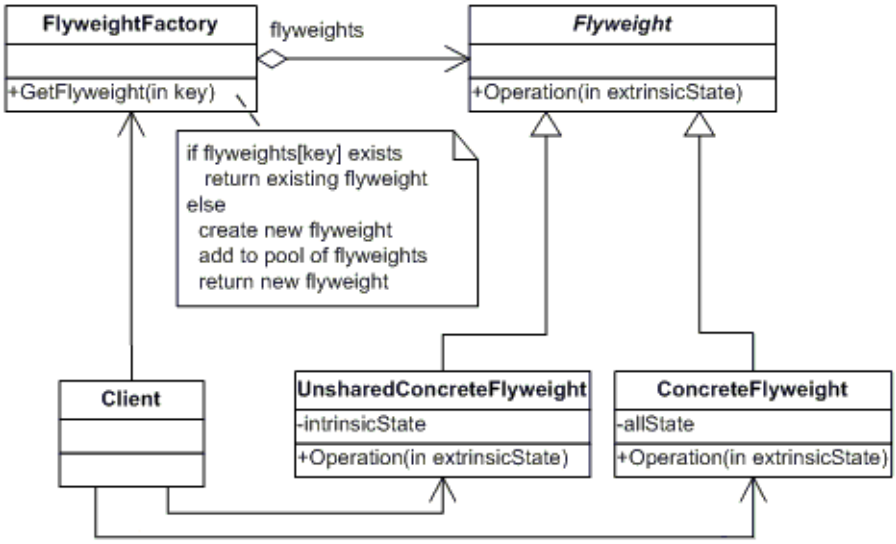
- Facade

  Estructura un entorno de programación y reduce su complejidad con la división en subsistemas, minimizando las comunicaciones y dependencias entre estos.



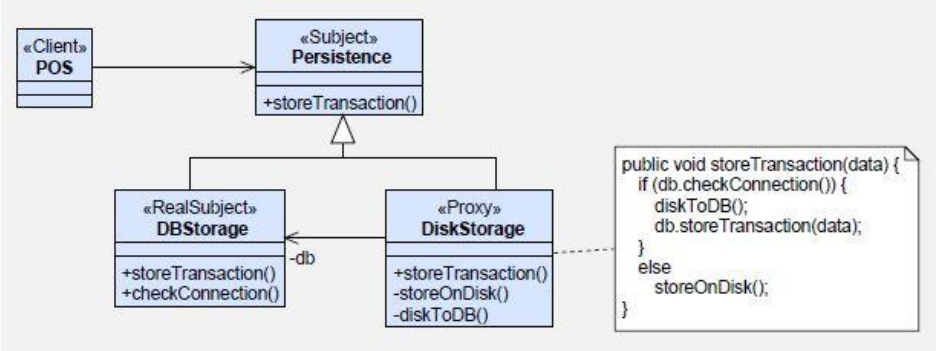
- Flyweight

  Uso compartido para admitir una gran cantidad de objetos que tienen parte de su estado interno en común, donde la otra parte del estado puede variar. Elimina o reduce la redundancia cuando tenemos gran cantidad de objetos que contienen información idéntica, además de lograr un equilibrio entre flexibilidad y rendimiento.



- Proxy

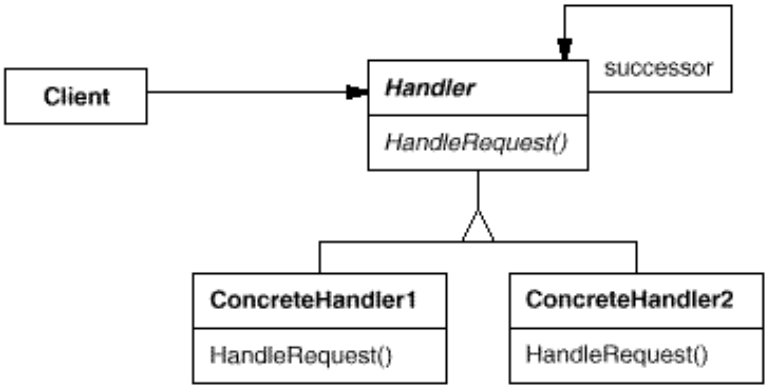
  Proporcionar un intermediario (placeholder) para que un objeto controle las referencias a él.



3. De Comportamiento

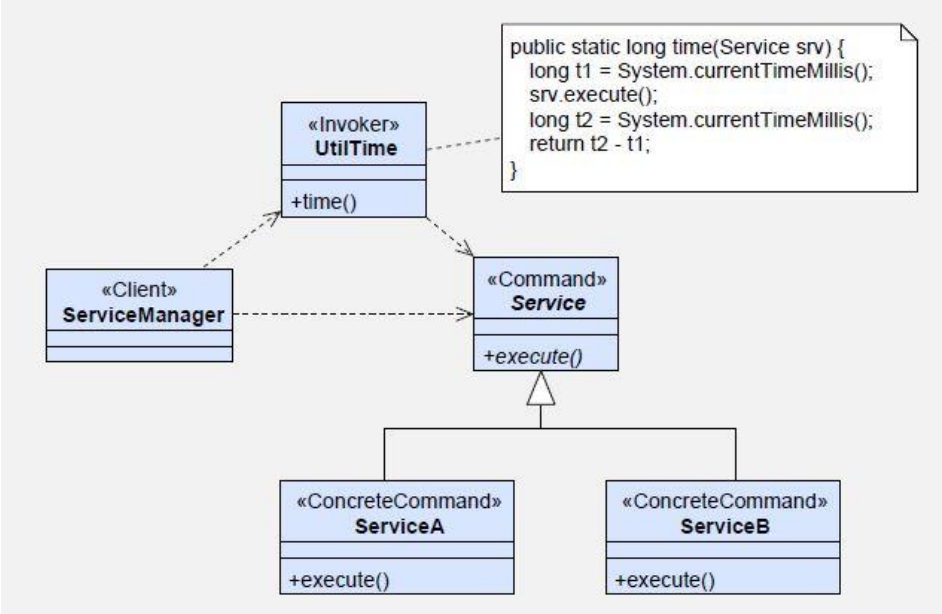
- Chain of Responsibility

  Evita acoplar el emisor de una petición a su receptor dando a más de un objeto la posibilidad de responder a una petición. Los objetos se convierten en partes de una cadena y la solicitud se envía de un objeto a otro a través de la cadena hasta que uno de los objetos lo maneje.



- Command

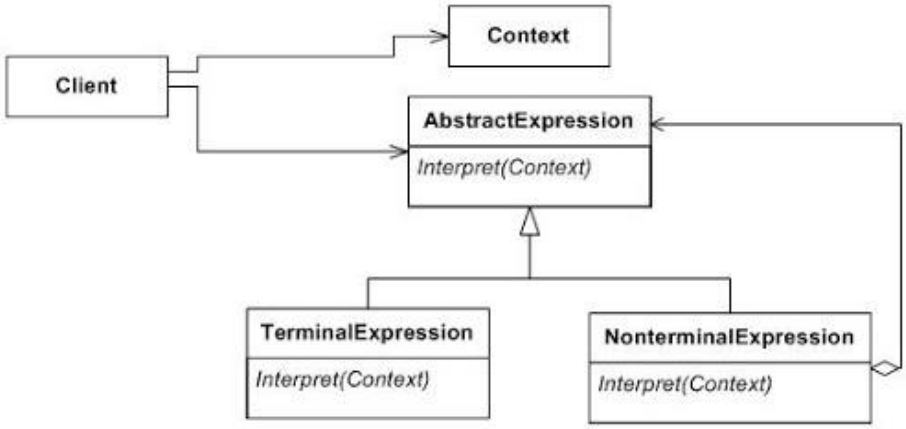
  Encapsula una solicitud en un objeto, permite la parametrización de clientes con diferentes solicitudes y permite guardar las solicitudes en una cola. Delega la ejecución de los servicios, de manera que ahora será una clase (Invoker) la encargada de ejecutarlos. Para esto es necesario convertir los servicios (operaciones) en objetos, manteniendo su inicialización donde se hacía originalmente, y añadiéndoles un método que permita ejecutarlos.



- Interpreter

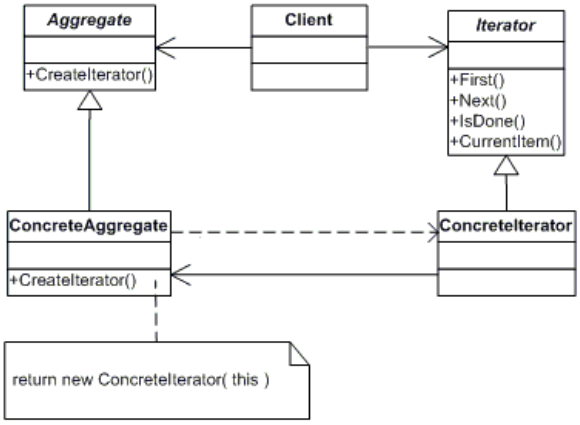
  Dado un idioma, define una representación para su gramática junto con un intérprete que use la representación para interpretar oraciones en el lenguaje. Asigna un dominio a un idioma, el lenguaje a una gramática y la gramática a un diseño jerárquico orientado a objetos.

  Uso: Se usa para definir un lenguaje para representar expresiones regulares que representen cadenas a buscar dentro de otras cadenas. Además, en general, para definir un lenguaje que permita representar las distintas instancias de una familia de problemas.



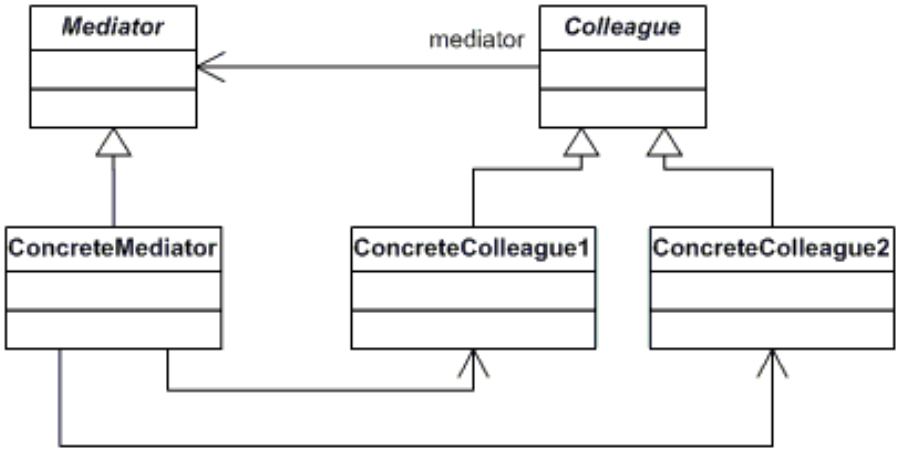
- Iterator

  Define una interfaz que declara los métodos necesarios para acceder secuencialmente a un grupo de objetos de una colección.



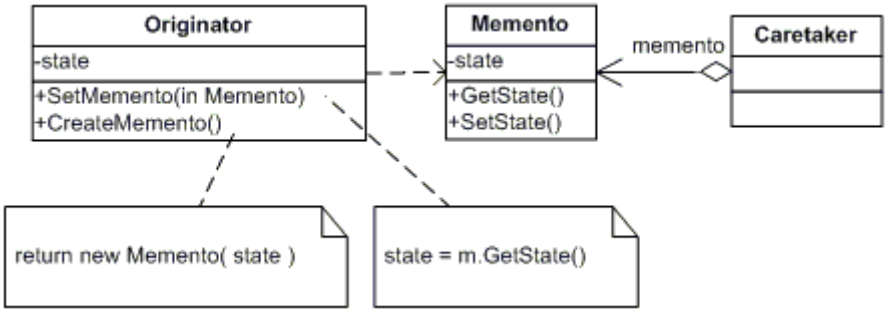
- Mediator

  Define un objeto que encapsula cómo interactúan un conjunto de objetos. El mediador promueve el acoplamiento flexible evitando que los objetos se refieran entre sí de forma explícita, y les permite variar su interacción de forma independiente.



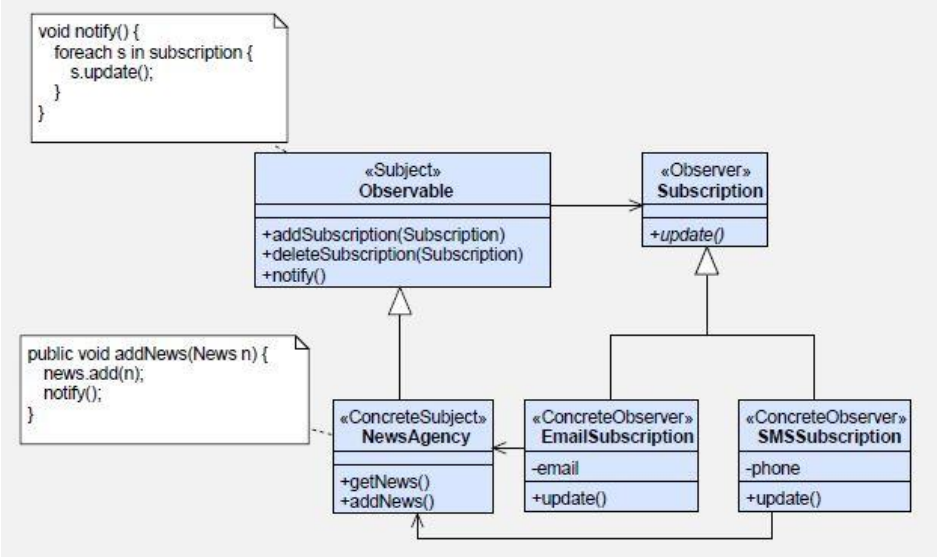
- Memento

  Capturar el estado interno de un objeto sin violar la encapsulación y así proporcionar un medio para restaurar el objeto al estado inicial cuando sea necesario.



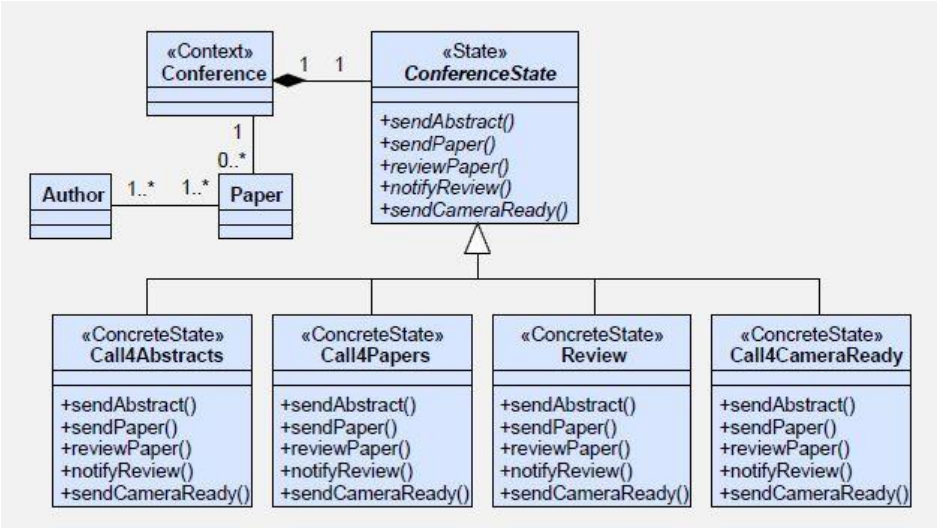
- Observer

  Define una dependencia de uno a muchos entre los objetos para que cuando un objeto cambie de estado, todos sus dependientes sean notificados y actualizados automáticamente.



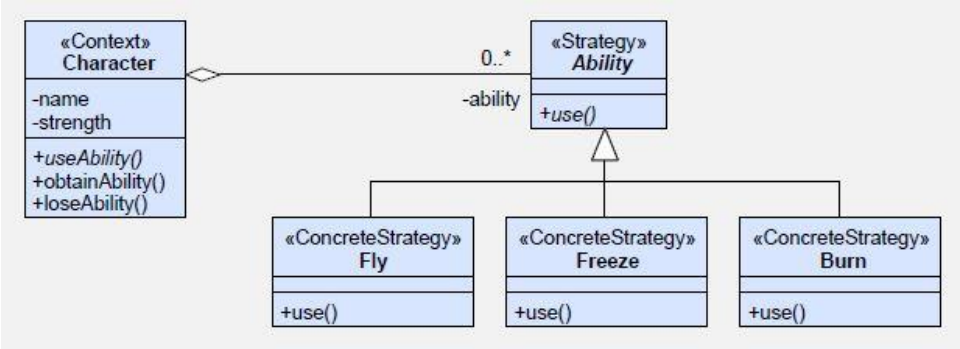
- State

  Se utiliza cuando el comportamiento de un objeto cambia dependiendo del estado del mismo.



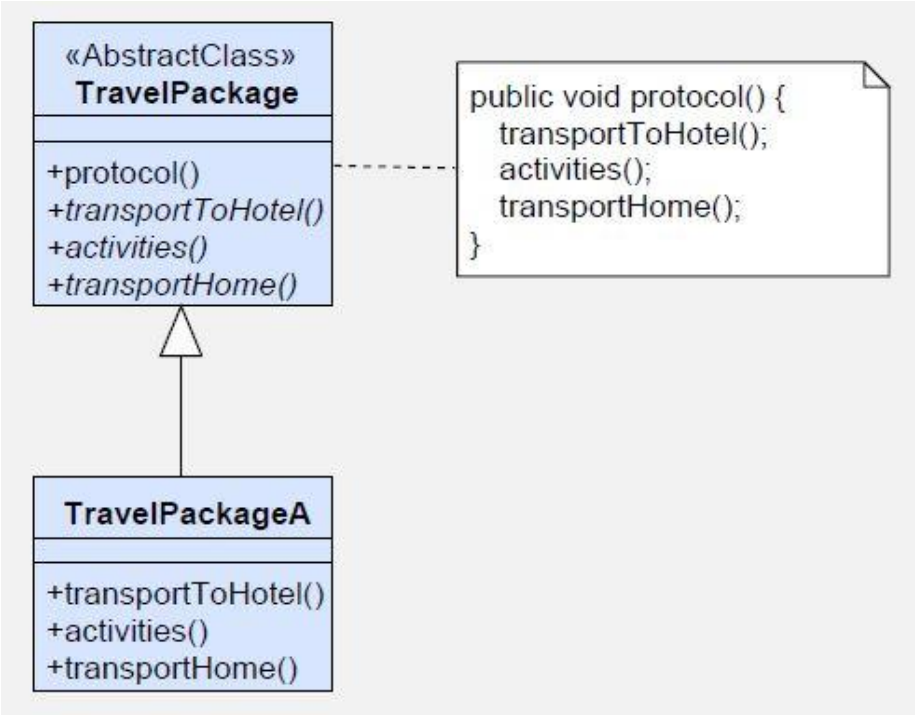
- ## Strategy

  Define una jerarquía de clases que representan algoritmos, encapsula cada una y hace que sean intercambiables. Estos algoritmos pueden ser intercambiados por la aplicación en tiempo de ejecución. Strategy permite que el algoritmo varíe independientemente de los clientes que lo usan.



- ## Template Method

  Define el esqueleto de un algoritmo en una operación, difiriendo algunos pasos a subclases. Template Method permite a las subclases redefinir ciertos pasos de un algoritmo sin permitirles cambiar la estructura del algoritmo.



- Vistor

  Representa una operación que debe realizarse en los elementos de una estructura de objeto. Visitor permite definir una nueva operación sin cambiar las clases de los elementos en los que opera.