Patrones GOF de comportamiento

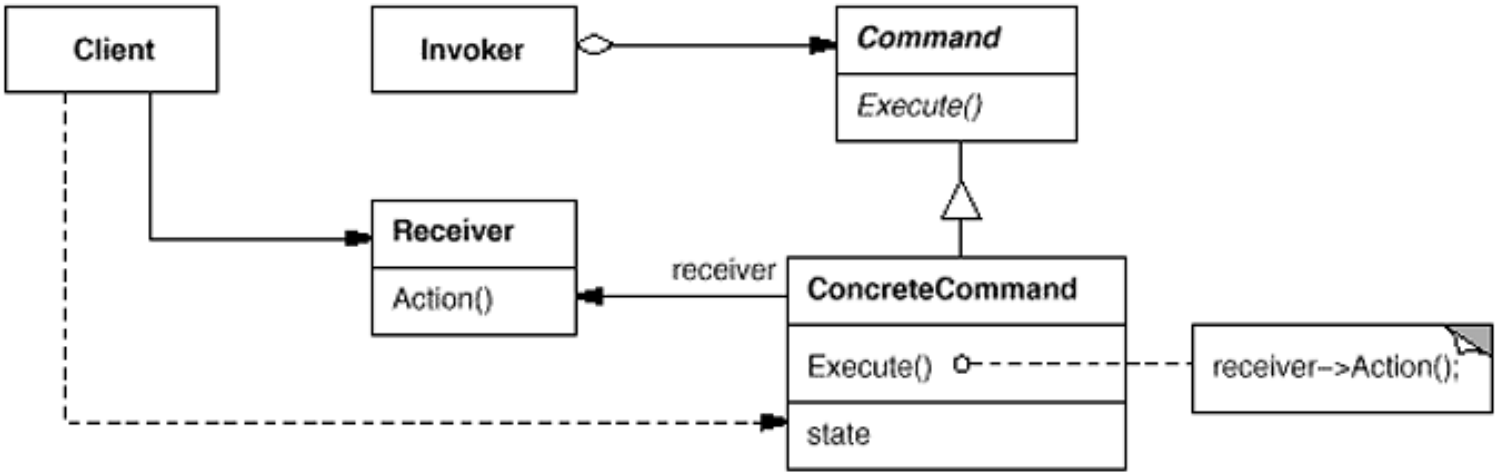
Diseño de Sistemas Software

Patrones GOF de comportamiento

- Los patrones de comportamiento tratan sobre algoritmos y la asignación de responsabilidades entre objetos. Además de las relaciones entre objetos o clases, describen también patrones de comunicación entre ellas.
  - Los **patrones de comportamiento de clases** usan la herencia para distribuir el comportamiento entre clases.
  - Los **patrones de comportamiento de objetos** usan la composición de objetos en lugar de la herencia.

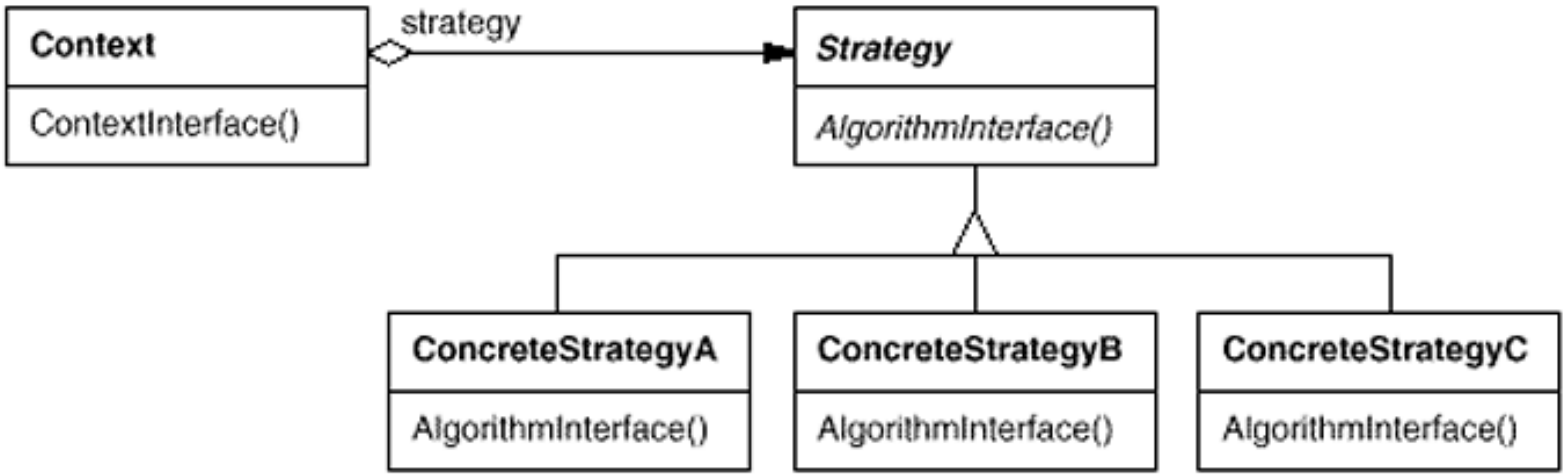
Command

- **Command**: representa una solicitud con un objeto, de manera tal de poder parametrizar a los clientes con distintas solicitudes, encolarlas o llevar un registro de las mismas, y poder deshacer las operaciones. Estas solicitudes, al ser representadas como un objeto también pueden pasarse como parámetro o devolverse como resultados.



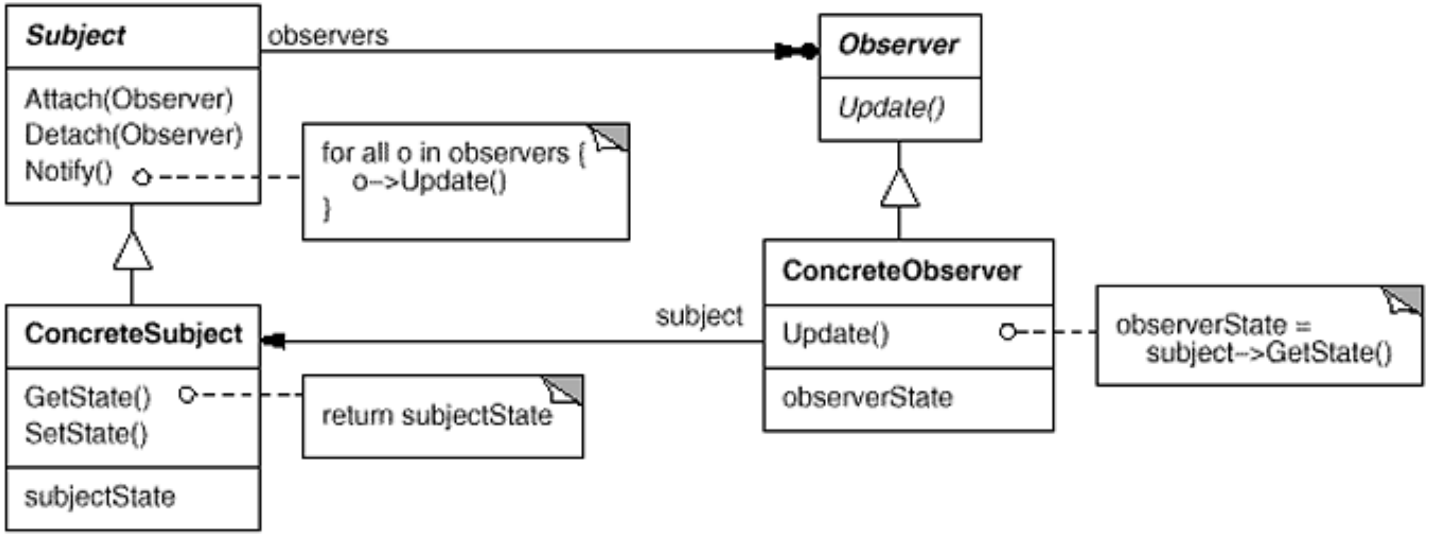
Strategy

- **Strategy**: define una jerarquía de clases que representan algoritmos, los cuales son intercambiables. Estos algoritmos pueden ser intercambiados por la aplicación en tiempo de ejecución.



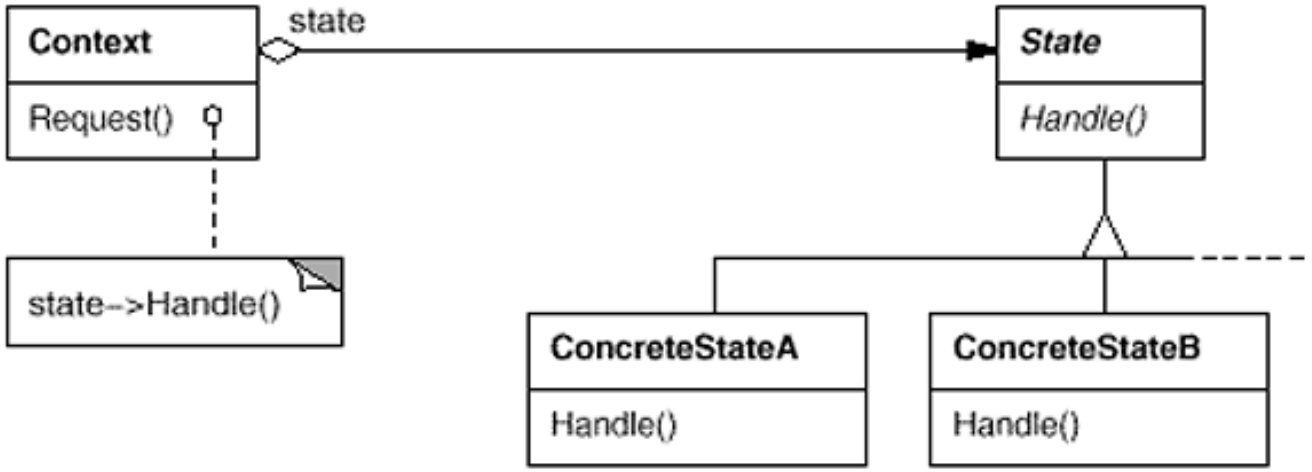
Observer

- **Observer**: brinda un mecanismo que permite a un componente transmitir de forma flexible mensajes a aquellos objetos que hayan expresado interés en él. Estos mensajes se disparan cuando el objeto ha sido actualizado, y la idea es que quienes hayan expresado interés reaccionen ante este evento.



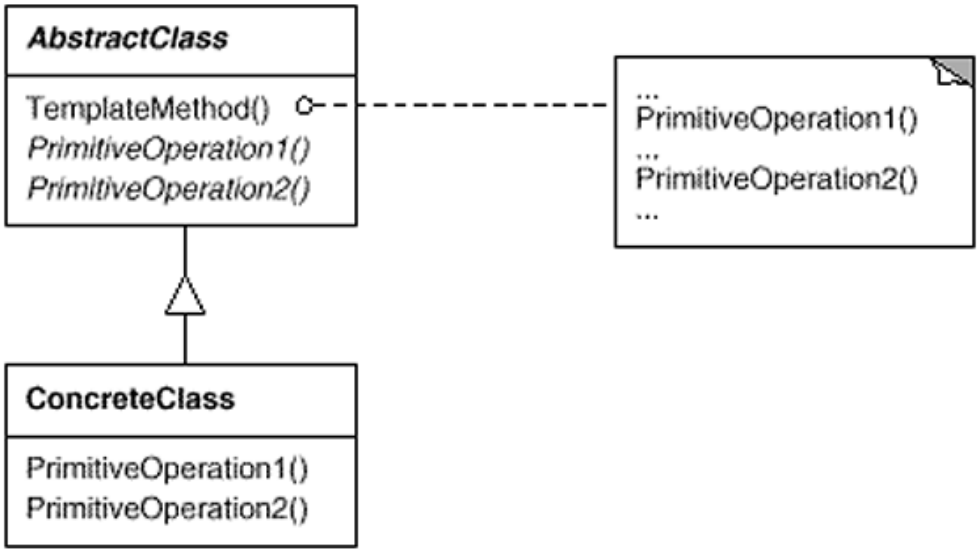
State

- **State**: permite que un objeto modifique su comportamiento cada vez que cambie su estado interno. El objeto parecerá que cambió de clase.



Template method

- **Template method**: define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura.



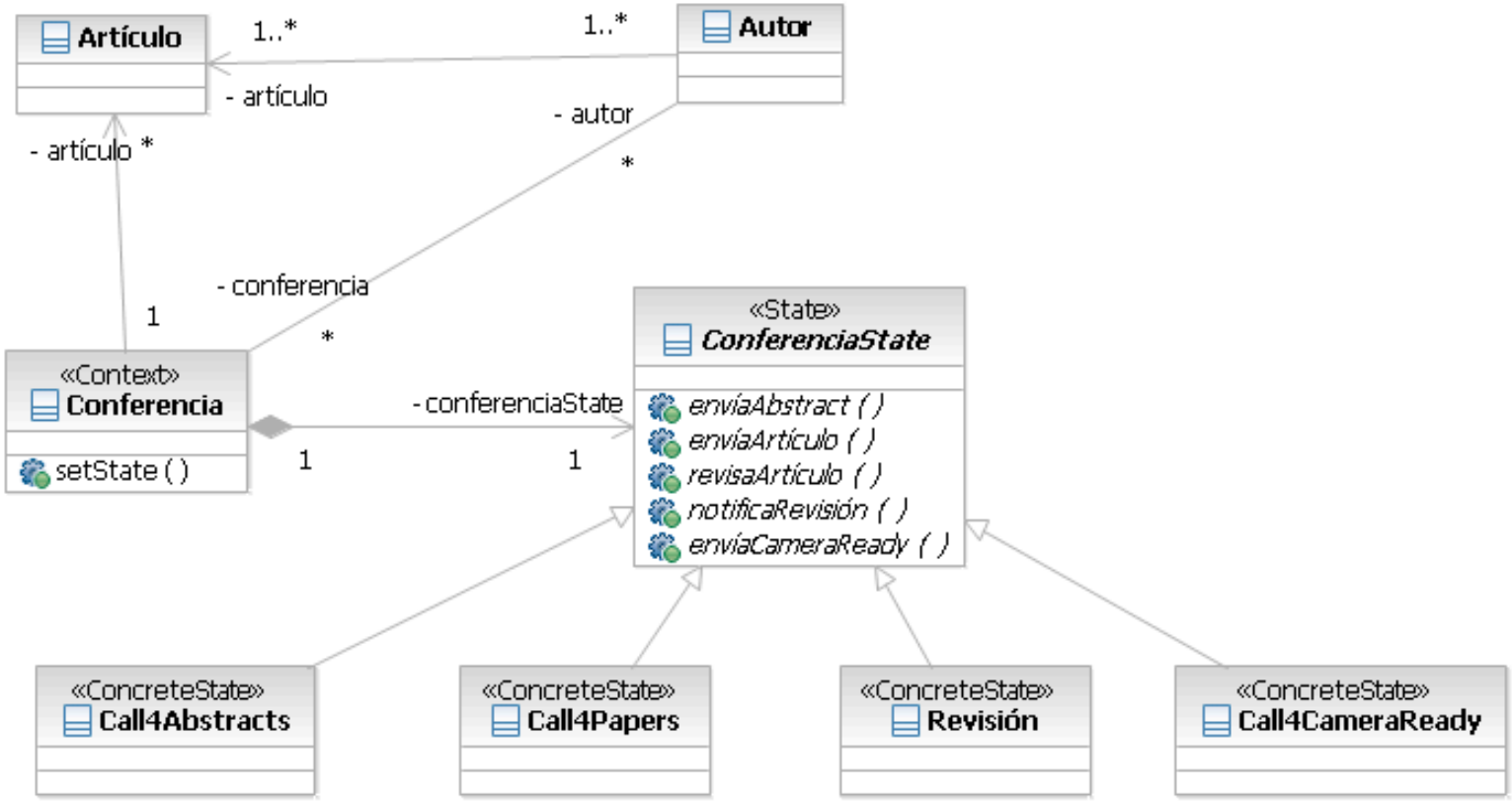
Ejercicios

Ejercicio 1

- **Problema**: Nos han pedido implementar un sistema para manejar el envío de artículos a conferencias. Una conferencia tiene distintas fases; por simplicidad vamos a asumir cuatro: Call4Abstracts, Call4Papers, Revisión, Call4CameraReady. El diseño inicial contiene tres clases: autor, artículo y conferencia. El sistema debe soportar cinco llamadas de interfaz: enviaAbstract(), enviaArtículo(), revisaArtículo(), notificaRevisión(), enviaCameraReady(). El comportamiento de estas operaciones variará según la fase del proceso de revisión en que nos encontremos. Plantea una solución a este problema.

Ejercicio 1

- **Solución**: patrón State

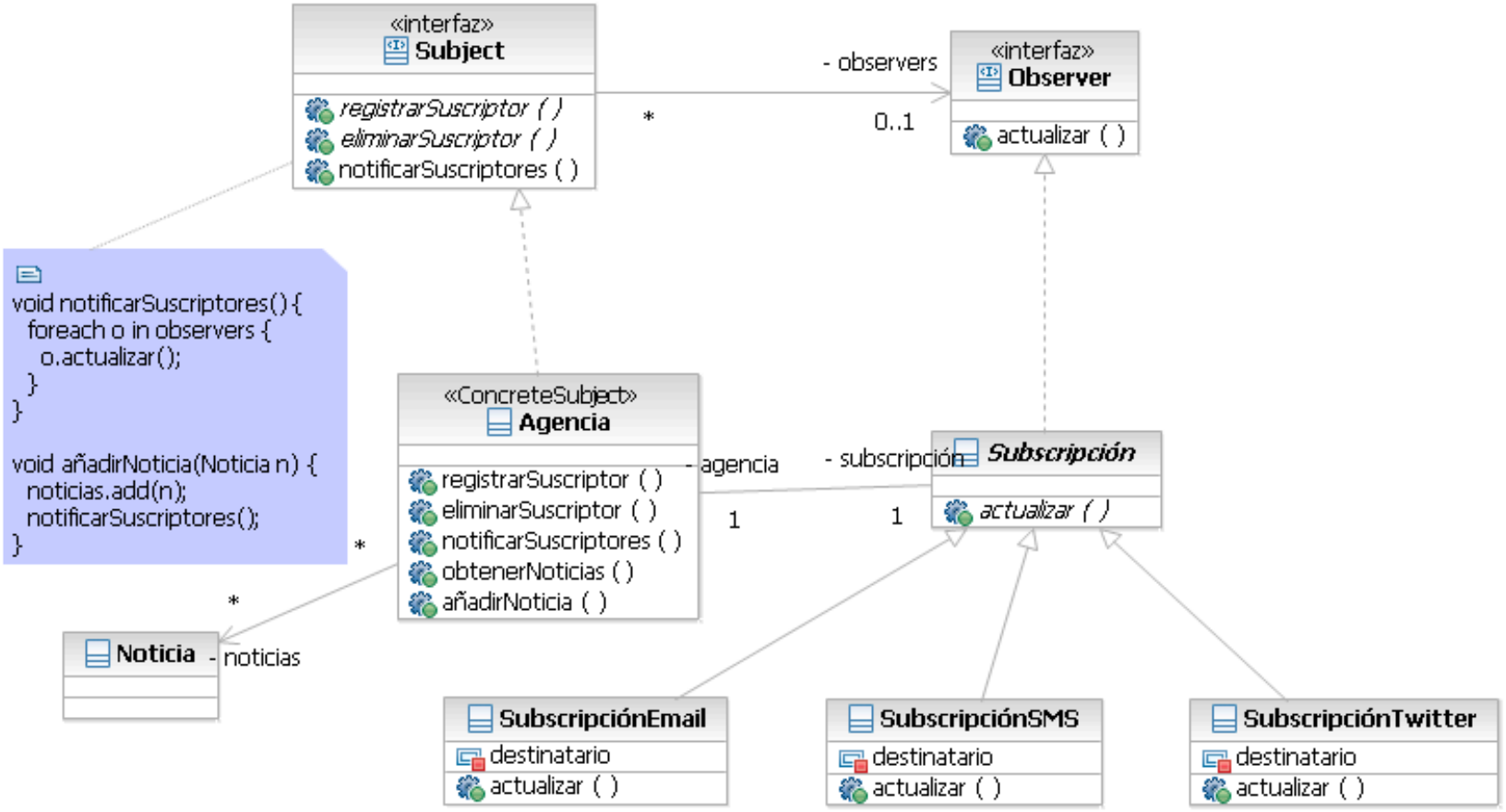


Ejercicio 2

- **Problema**: Nos ha contactado la Agencia F de noticias. Esta agencia recoge noticias y comunicados de todo el mundo, y las distribuye a todos sus clientes (periódicos, cadenas de televisión, etc.). Nos han pedido que creemos un framework para que la agencia pueda informar inmediatamente, cuando ocurre cualquier evento, a cualquiera de sus clientes. Estos clientes pueden recibir las noticias de distintas maneras: Email, SMS, etc. Además, nos han pedido que la solución sea lo suficientemente extensible para soportar nuevos tipos de subscripciones (e.g. recientemente ya hay algún cliente que les ha pedido ser notificado a través de Twitter).

Ejercicio 2

- **Solución**: patrón Observer

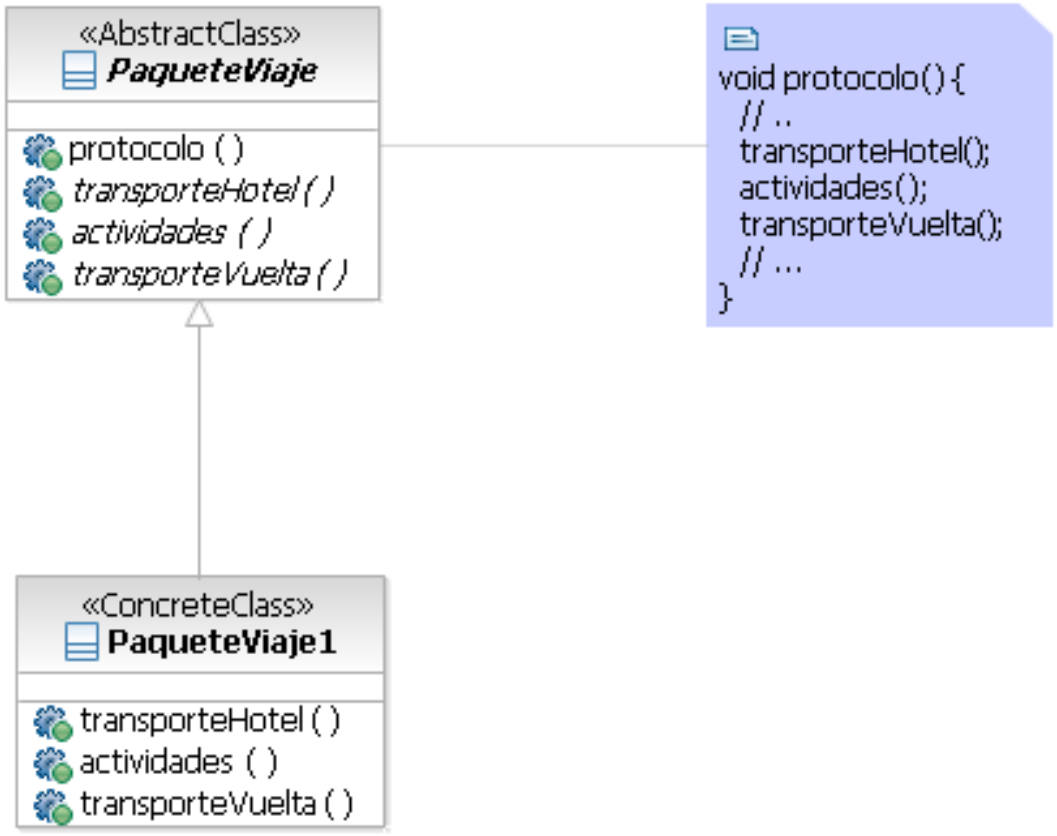


Ejercicio 3

- **Problema**: Nos han pedido implementar una aplicación para una agencia de viajes. La agencia maneja distintos tipos de paquetes; todos ellos comparten un protocolo común, que incluye lo siguiente:
  - Todos los turistas son transportados al lugar de destino por el medio de transporte seleccionado
  - Cada día tienen una actividad programada
  - Todos los turistas vuelven (si quieren ☺)
- Sin embargo, según el paquete elegido, hay ciertas variaciones:
  - Los turistas pueden ser llevados en transfer desde el hotel al lugar donde cojan su transporte o ir por su cuenta.
  - Las actividades programadas pueden estar incluidas o no en el precio del paquete
  - …

Ejercicio 3

- **Solución**: patrón Template Method



Ejercicio 4

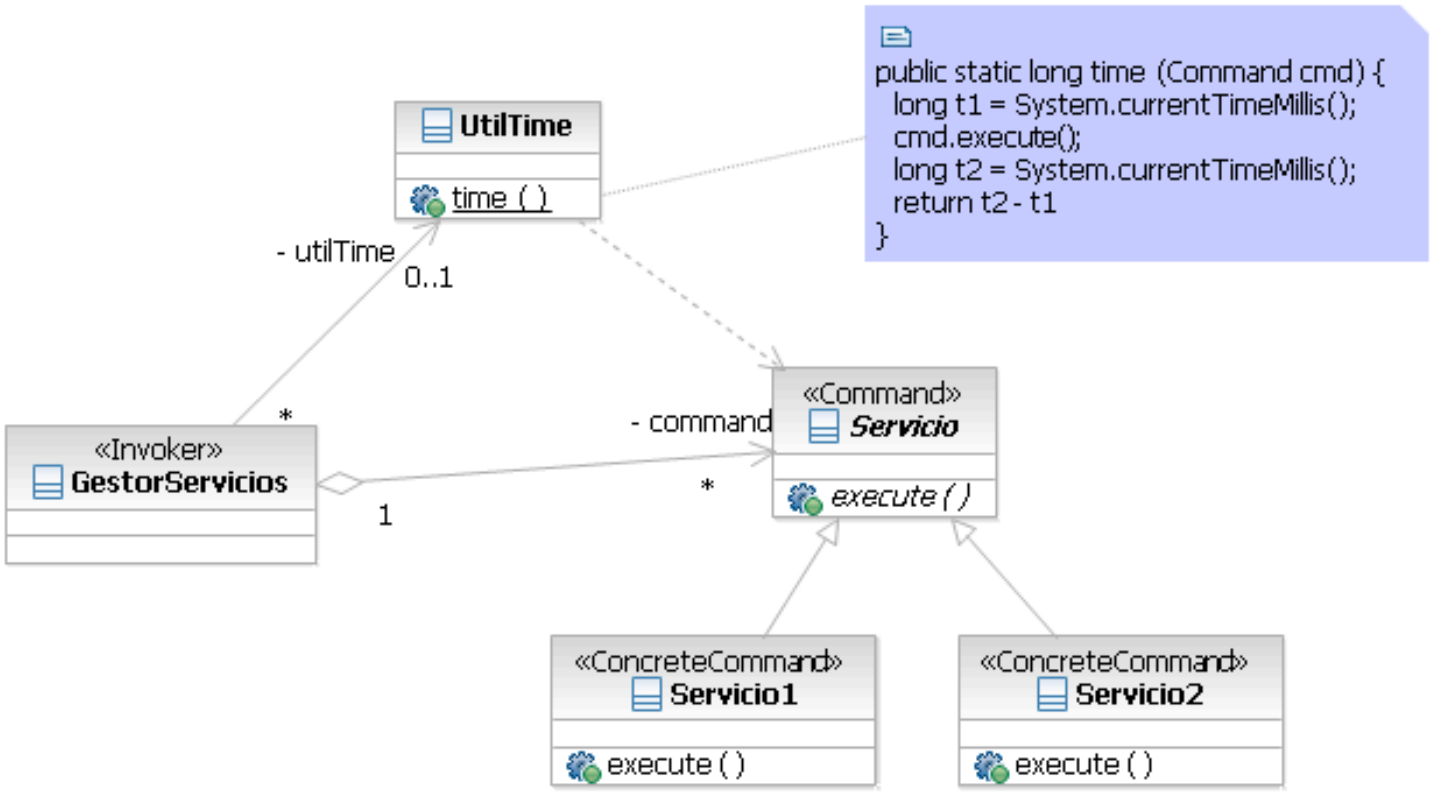
- **Problema**: En un sistema que ofrece servicios a clientes remotos se desea realizar una auditoría del tiempo que tardan los distintos servicios en ejecutarse, a fin de mejorar el rendimiento global del sistema. Para esto se ha implementado una clase UtilTime, que no es abstracta, que incluye un método estático time que mide el tiempo que tarda un método cualquiera en ejecutarse. Parte del código de dicho método sería:

```
public static long time ( “parámetros”) {
          long t1 = System.currentTimeMillis();

          // falta aquí el código apropiado

          long t2 = System.currentTimeMillis();
          return t2 – t1
}
```

- Diseña una solución basada en alguno de los patrones GOF y completa el método: parámetros y código en la posición del comentario. Escribe un código cliente con un ejemplo de utilización del método time().

Ejercicio 4

- **Solución**: patrón Command

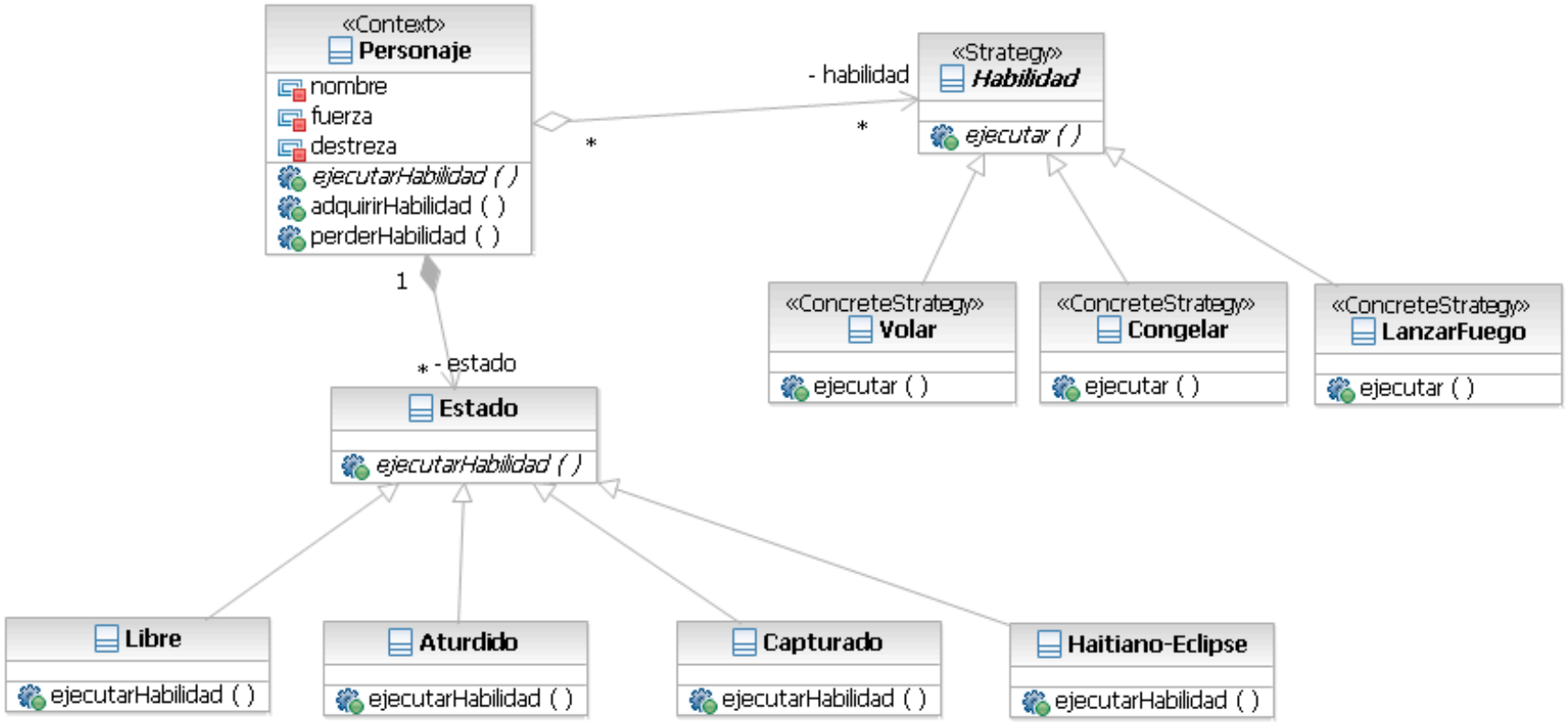


Ejercicio 5

- **Problema**: Suponed que estamos desarrollando un juego interactivo basado en la serie *Héroes*. El juego cuenta con distintos personajes, y cada uno tiene una serie de habilidades especiales: volar, lanzar rayos, leer la mente, regenerarse, teletransportarse, … Los personajes pueden adquirir/perder poderes de forma dinámica según se desarrolla la trama del juego. Por ejemplo, el padre de Peter y Nathan Petrelli puede desposeer a cualquier personaje de sus poderes. Sylar, por otro lado, va adquiriendo poderes según va analizando a sus víctimas. Además, cualquier personaje puede ver inhabilitados sus poderes cuando están capturados o cuando el Haitiano está cerca o hay un eclipse. Los personajes también ven modificado el comportamiento de sus habilidades cuando están aturdidos.

Ejercicio 5

- **Solución**: patrón Strategy



Bibliography

- *Design Patterns: Elements of Reusable Object-Oriented Software*. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Addison-Wesley Professional, 1994.