|  |  |  |  |
| --- | --- | --- | --- |
| NOMBRE | Diseño de sistemas | | |
| AREA | Tecnologías Aplicadas | REGIMEN | Semestral |
| MODALIDAD | Por examen final | CARGA HORARIA | |  | | --- | | 6 (seis) semanales,  90 totales | |

Se entrega la práctica en formato digital y papel. El formato digital debe tener el mismo encabezado que el presente texto. El formato en papel debe contener fecha, materia, curso, datos del alumno.

***Alumno:***

***Curso:***

***Fecha de entrega:***

**Tema: Patrones de diseño**

**Mediator**

Un Mediator es un patrón de diseño que define un objeto que hace de procesador central, coordinando las relaciones entre sus asociados o participantes. Permite la interacción de varios objetos, sin generar acoples fuertes en esas relaciones. Todos los objetos se comunican con un mediador y es éste quién realiza la comunicación con el resto.

Cuando muchos objetos interactúan con otros objetos, se puede formar una estructura muy compleja, con muchas conexiones entre distintos objetos. En un caso extremo cada objeto puede conocer a todos los demás objetos. Para evitar esto, el patrón Mediator, encapsula el comportamiento de todo un conjunto de objetos en un solo objeto.

Usar el patrón Mediator cuando:

Un conjunto grande de objetos se comunica de una forma bien definida, pero compleja.

Reutilizar un objeto se hace difícil porque se relaciona con muchos objetos.

Las clases son difíciles de reutilizar porque su función básica esta entrelazada con relaciones de dependencia.

Un ejemplo real que se puede comparar con este patrón es un framework que se llama Struts. Struts posee una clase que hace de Mediadora que se llama ActionServlet. Esta clase lee un archivo de configuración (el struts-config.xml) para ayudarse con la mediación, pero lo importante es que se encarga de comunicar el flujo de información de todos los componentes web de una aplicación. Si no utilizamos Struts, entonces cada página debe saber hacia dónde debe dirigir el flujo y el mantenimiento de dicha aplicación puede resultar complicado, especialmente si la aplicación es muy grande.

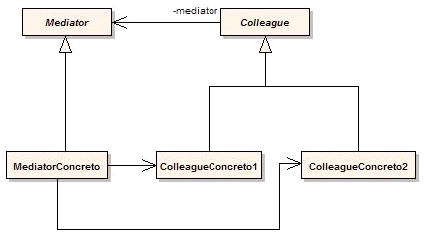
En cambio, si todas las páginas se comunican con un mediador, entonces la aplicación es mucho más robusta: con cambiar un atributo del mediador todo sigue funcionando igual.

Como conclusión podemos afirmar que este patrón debe ser utilizado en casos donde convenga utilizar un procesador central, en vez de que cada objeto tenga que conocer la implementación de otro. Imaginemos un aeropuerto: que pasaría si no tuviese una torre de control y todos los aviones que deban aterrizar/despegar se tienen que poner todos de acuerdo para hacerlo. Además cada avión debe conocer detalles de otros aviones (velocidad de despegue, nafta que le queda a cada uno que quiera aterrizar, etc).

Para evitar esto se utiliza una torre de control que sincroniza el funcionamiento de un aeropuerto. Esta torre de control se puede ver como un mediador entre aviones.

**Diagramas UML**

**Vista estática: diagrama de clases**



Mediator: define una interface para comunicarse con los objetos colegas.

MediatorConcreto: implementa la interface y define como los colegas se comunican entre ellos. Además los conoce y mantiene, con lo cual hace de procesador central de todos ellos.

Colleague: define el comportamiento que debe implementar cada colega para poder comunicarse el mediador de una manera estandarizada para todos.

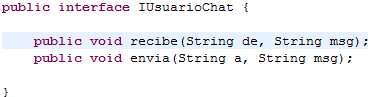
ColleagueConcreto: cada colega conoce su mediador, y lo usa para comunicarse con otros colegas.

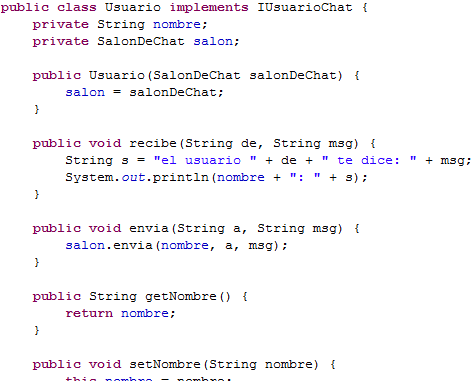
Los colegas envían y reciben requerimientos de un objeto mediador. El mediador gestiona cada mensaje y se lo comunica a otro colega si fuese necesario.

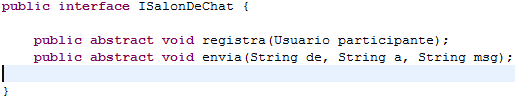
**Vista Dinámica: Diagrama de secuencia**

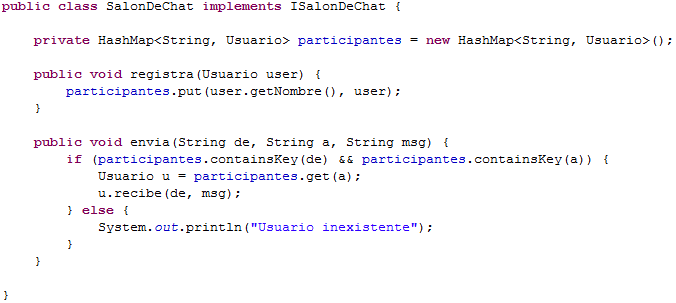
***Caso práctico a desarrollar en clase***

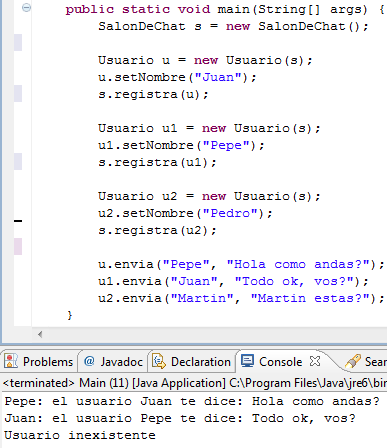
Nuestro ejemplo será un chat: donde habrá usuarios que se comunicaran entre sí en un salón de chat. Para ellos se define una interface llamada IUsuarioChat que todos los objetos que quieran participar de un chat deberán implementar. La clase Usuario representa un usuario que quiera chatear.











Consecuencias

No es necesario exponer el estado interno como atributos de acceso público, preservando así la encapsulación.

Si el originador tendría que de almacenar y mantener a salvo una o muchas copias de su estado interno, sus responsabilidades crecerían y sería inmanejable.

El uso frecuente de Mementos para almacenar estados internos de gran tamaño, podría resultar costoso y perjudicar la performance del sistema.

Caretaker no puede hacer predicciones de tiempo ni de espacio.

**Temas a tener en cuenta.**

Si bien la implementación de un Memento no suele variar demasiado, cuando la secuencia de creación y restauración de mementos es conocida, se puede adoptar una estrategia de cambio incremental: en cada nuevo memento sólo se almacena la parte del estado que ha cambiado en lugar del estado completo.

Esta estrategia se aplica cuando memento se utiliza para mantener una lista de deshacer/rehacer.

Otra opción utilizada es no depender de índices en la colecciones y utilizar ciertos métodos no indexados como el .previus () que poseen algunas colecciones.