**UNIVERSIDAD AUTONOMA GABRIEL RENE MORENO**

**FACULTAD CIENCIAS DE LA COMPUTACION Y TELECOMUNICACIONES**

****

**MATERIA : ARQUITECTURA DE SOFTWARE**

**DOCENTE : ING. VEIZAGA**

**FECHA : 29/11/2019**

**INTEGRANTES:**

**ROJAS VALDEZ GUSTAVO RENE**

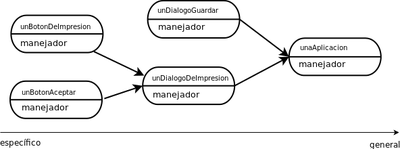
Santa Cruz- Bolivia

# Cadena de responsabilidad

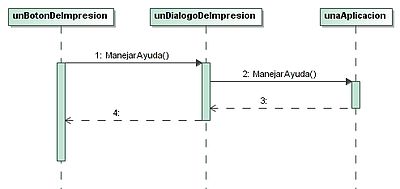
El [patrón de diseño](https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o) **Chain of Responsibility** es un patrón de comportamiento que evita acoplar el emisor de una petición a su receptor dando a más de un objeto la posibilidad de responder a una petición. Para ello, se encadenan los receptores y pasa la petición a través de la cadena hasta que es procesada por algún objeto. Este patrón es utilizado a menudo en el contexto de las interfaces gráficas de usuario donde un objeto puede estar compuesto de varios objetos (que generalmente heredan de una super clase "vista"). No se debe confundir con el patrón [Composite (patrón de diseño)](https://es.wikipedia.org/wiki/Composite_(patr%C3%B3n_de_dise%C3%B1o)" \o "Composite (patrón de diseño)) que se basa en un concepto similar. Según si el ambiente de ventanas genera eventos, los objetos los manejan o los "pasan" (transmiten) hasta que algún elemento consume dicho evento y se detiene la propagación. Un claro ejemplo de cadena de responsabilidades sería una estructura de datos que contiene nodos cuya implementación es opaca para el usuario de dicha estructura de datos (no conoce salvo la interfaz de la estructura de datos). Internamente los nodos se relacionan mediante punteros y una llamada a un método de la estructura de datos hará que un mensaje se propague por la cadena de nodos hasta llegar a su receptor. Es decir, una petición de "búsqueda(clave)" al contenedor de nodos hará que el mismo pase dicho mensaje a uno de los nodos (raíz, primero, etc; dependiendo de como se haya implementado la estructura de datos) devolviendo el nodo el resultado si su atributo clave coincide con la búsqueda o mandando dicho mensaje al siguiente nodo en caso contrario. Obviamente se daría este caso a niveles teóricos pues existen formas más optimas de solventar el anterior problema de ejemplo y; sobre todo, ha de tenerse en cuenta que una cadena muy grande de muchos elementos puede llegar a apilar en memoria una gran cantidad de llamadas a procedimientos hasta que la cadena retorne un resultado con su consiguiente coste asociado.

## Motivación

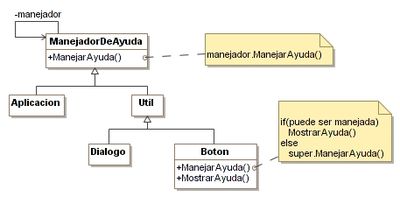
Supongamos un servicio de ayuda sensible al contexto para una interfaz gráfica. El usuario puede obtener ayuda en cualquier parte de la interfaz pulsando con el ratón sobre ella. La ayuda proporcionada depende de la parte de la interfaz que se haya seleccionado así como de su contexto. Si no existe información de ayuda específica para esa parte el sistema debería mostrar un mensaje de ayuda más general sobre el contexto inmediato. El problema es que el objeto que en última instancia proporciona la ayuda no conoce explícitamente al objeto (por ejemplo, el botón) que inicializa la petición. Necesitamos un modo de desacoplar el botón que da lugar a la petición de ayuda de los objetos que podrían proporcionar dicha información. El patrón Cadena de Responsabilidad define cómo hacerlo. La idea de este patrón es desacoplar a los emisores y a los receptores dándole a varios objetos la posibilidad de tratar una petición. La petición se pasa a través de una cadena de objetos hasta que es procesada por uno de ellos.

[](https://commons.wikimedia.org/wiki/File:Motivacion2ES.png)

Supongamos que el usuario solicita ayuda sobre un botón denominado "Imprimir", el cual se encuentra en una instancia de Dialogo De Impresión. El siguiente diagrama muestra cómo la petición de ayuda se reenvía

[](https://commons.wikimedia.org/wiki/File:Motivacion3ES.jpg)

En este caso, la petición no es procesada ni por un Boton De Impresion ni por un Dialogo De Impresion; se detiene en una Aplicación que podrá procesarla u obviarla. El cliente que dio origen a la petición no tiene ninguna referencia directa al objeto que finalmente la satisface. Para reenviar la petición a lo largo de la cadena, y garantizar que los receptores permanecen implícitos, cada objeto de la cadena comparte una interfaz común para procesar peticiones y para acceder a su sucesor. Por ejemplo, en este sistema de ayuda podría definirse una clase Manejador De Ayuda.

[](https://commons.wikimedia.org/wiki/File:Motivacion1ES.jpg)

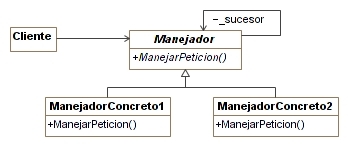
Las clases Botón, Dialogo y Aplicación usan las operaciones de Manejador De Ayuda para tratar peticiones de ayuda. La operación Manejar Ayuda de Manejador De Ayuda reenvía la petición al sucesor de manera predeterminada. Las subclases pueden redefinir esta operación para proporcionar ayuda en determinadas circunstancias; en caso contrario, pueden usar la implementación predeterminada para reenviar la petición.

## Aplicabilidad

El patrón Cadena de Responsabilidad debe usarse cuando:

* hay más de un objeto que puede manejar una petición, y el manejador no se conoce a priori, sino que debería determinarse automáticamente.
* se quiere enviar una petición a un objeto entre varios sin especificar explícitamente el receptor.
* el conjunto de objetos que pueden tratar una petición debería ser especificado dinámicamente.

## Estructura

[](https://commons.wikimedia.org/wiki/File:EstructuraES2.jpg)

## Participantes

* **Manejador**: define una interfaz para tratar las peticiones. Opcionalmente, implementa el enlace al sucesor.
* **Manejador Concreto**: trata las peticiones de las que es responsable; si el Manejador Concreto puede manejar la petición, lo hace; en caso contrario la reenvía a su sucesor.
* **Cliente**: inicializa la petición a un Manejador Concreto de la cadena.

## Colaboraciones

Cuando un cliente envía una petición, ésta se propaga a través de la cadena hasta que un objeto Manejador Concreto se hace responsable de procesarla.

## Ventajas

Las ventajas de este patrón son:

* **Reduce el acoplamiento**. El patrón libera a un objeto de tener que saber qué otro objeto maneja una petición. Ni el receptor ni el emisor se conocen explícitamente entre ellos, y un objeto de la cadena tampoco tiene que conocer la estructura de ésta. Por lo tanto, simplifica las interconexiones entre objetos. En vez de que los objetos mantengan referencias a todos los posibles receptores, sólo tienen una única referencia a su sucesor.
* **Añade flexibilidad para asignar responsabilidades a objetos**. Se pueden añadir o cambiar responsabilidades entre objetos para tratar una petición modificando la cadena de ejecución en tiempo de ejecución. Esto se puede combinar con la herencia para especializar los manejadores estáticamente.

Por otra parte presenta el inconveniente de no garantizar la recepción. Dado que las peticiones no tienen un receptor explícito, no hay garantías de que sean manejadas. La petición puede alcanzar el final de la cadena sin haber sido procesada.

## Implementación

* **Implementación de la cadena sucesora**. Hay dos formas posibles de implementarla:

1. Definir nuevos enlaces (normalmente en el Manejador, pero también podría ser en los objetos Manejador Concreto).
2. Usar enlaces existentes (otras asociaciones existentes). Por ejemplo, en el patrón Composición puede existir ya que un enlace al padre puede utilizarse para definir la cadena de responsabilidad sin necesidad de añadir otra asociación.

* **Conexión de los sucesores**. Si no hay referencias preexistentes para definir una cadena, entonces tendremos que introducirlas nosotros mismos. En este caso, el Manejador define la interfaz y además, se encarga de mantener el sucesor. Esto permite que el manejador proporcione una implementación predeterminada de Manejar Petición que reenvíe la petición al sucesor (si hay alguno). Si una subclase de Manejador Concreto no está interesada en dicha petición, no tiene que redefinir la operación de reenvío.
* **Representación de peticiones**. Hay varias opciones para representar las peticiones:
* Una petición es una invocación a una operación insertada en el código. Esto resulta conveniente y seguro, pero sólo se pueden reenviar el conjunto prefijado de peticiones que define la clase Manejador.
* Una única función manejadora que reciba un código de petición como parámetro. Esto permite un número arbitrario de peticiones pero emisor y receptor deben ponerse de acuerdo sobre cómo codificarse la petición.

# Descripción del problema

Estamos realizando el software para un banco y uno de los puntos más importantes es saber quién puede aprobar un crédito. Por lo tanto el banco define las siguientes reglas de negocio:

Si el monto no supera los $ 10.000 entonces el ejecutivo de cuentas pueda aprobar el préstamo.

Si el monto esta entre los $10.000 y $50.000 entonces la persona indicada para realizar la aprobación es el Jefe inmediato de dicho ejecutivo.

Si el monto se encuentra entre $ 50.000 y $100.000 entonces es el Gerente quién debe realizar dicha aprobación.

Por montos superiores a los $100.000 entonces la aprobación la realizará el Director.

# Solución

## Diagrama de clases

