



**PATRÓN DE DISEÑO**

**TRANSACCIÓN**

ESTUDIANTE: Rivero Conde Daniel Gonzalo

REGISTRO: 212018231

TEMA: Patrones de Diseño

DOCENTE: Ing. Josué Obed Veizaga Gonzales

MATERIA: Arquitectura de Software

SEMESTRE: 2 / 2019

Contenido

[1. Descripción del patrón 2](#_Toc26154237)

[2. Clasificación 2](#_Toc26154238)

[3. Diagrama genérico 2](#_Toc26154239)

[4. Problema 3](#_Toc26154240)

[5. Diagrama del Problema 4](#_Toc26154241)

1. **Descripción del patrón**

Permite agrupar una colección de métodos para que todos puedan ser exitosos o fallar colectivamente.

El patrón se utiliza cuando:

* Cuando los métodos tienen que estar sincronizados.
* Cuando necesitemos asegurar que las transacciones se realizaron correctamente.

En la programación orientada a objetos, generalmente se trata de múltiples instancias de múltiples clases. A veces, sin embargo, debe tratar varios objetos como si fueran un solo objeto; al menos, debe asegurarse de que ese estado permanece constante entre los objetos a medida que se realiza una operación.

1. **Clasificación**

Este patrón corresponde al grupo de **Patrones de Sistema**.

1. **Diagrama genérico**



Los participantes del patrón son los siguientes:

* Transaction Participant: Es la interfaz que define los métodos para el control de cada participante.
* Specifc Participant: Es una extensión para la interfaz genérica, esta interfaz contiene los métodos de negocio.
* Concrete Participant: Este implementa la interfaz Specific Participant. Este define que está pasando si el cliente decide cancelar o si se produce un error o si todo se ejecuta correctamente. Tiene que mantener una referencia al estado original para poder restaurarlo cuando se invoca cancelar.

La secuencia normal para usar una transacción es:

1. Crear una ID de transacción (ya sea Objeto o Long).
2. Invoque la unión en todos los participantes y cancele la transacción si la unión falla para alguno de los participantes.
3. Pruebe la acción, invoque los métodos necesarios y llame a cancelar tan pronto como falle un participante.
4. Cuando se complete la acción, llame a commit en todos los participantes.

# **Problema**

Un ejemplo es transferir fondos de una cuenta a otra: Las tareas de retirar dinero de una cuenta y agregarlo a otra cuenta debe realizarse, o ninguno debe realizarse. El resultado final siempre debe estar en equilibrio: la cantidad deducida de la primera debe agregarse a la segunda y viceversa.

Si la deducción de la primera cuenta falla (la primera cuenta ya está por encima de su límite) la segunda cuenta no debe ser aumentado. Si simplemente confía en que cada operación es exitosa, transferir fondos sería muy fácil y muy arriesgado. Si esta transferencia falla, es posible que vea que los fondos desaparecen mágicamente de una de sus cuentas, algo malo de su cuenta perspectiva.

También puede ver que los fondos aparecen mágicamente en otra cuenta, algo bueno para usted, pero ellos bancos consideran esto una cosa muy mala.

1. **Diagrama del Problema**



1. **Bibliografía**

* Prentice Hall (2001), *Applied Java Patterns,* Prentice Hall PTR.