

# Ingeniería de Software II (CI-4713)

## Patrones

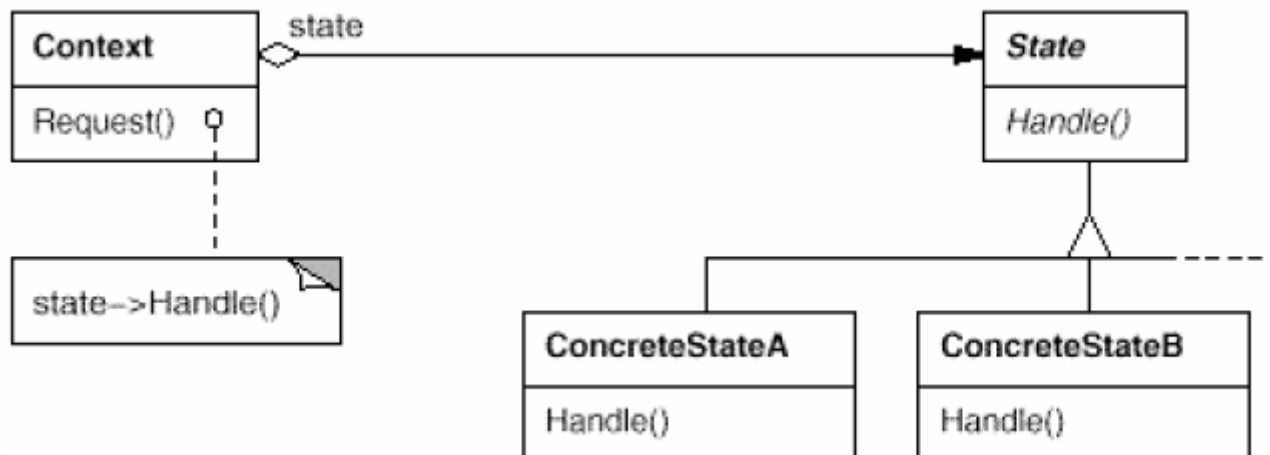
mayo de 2008

# State

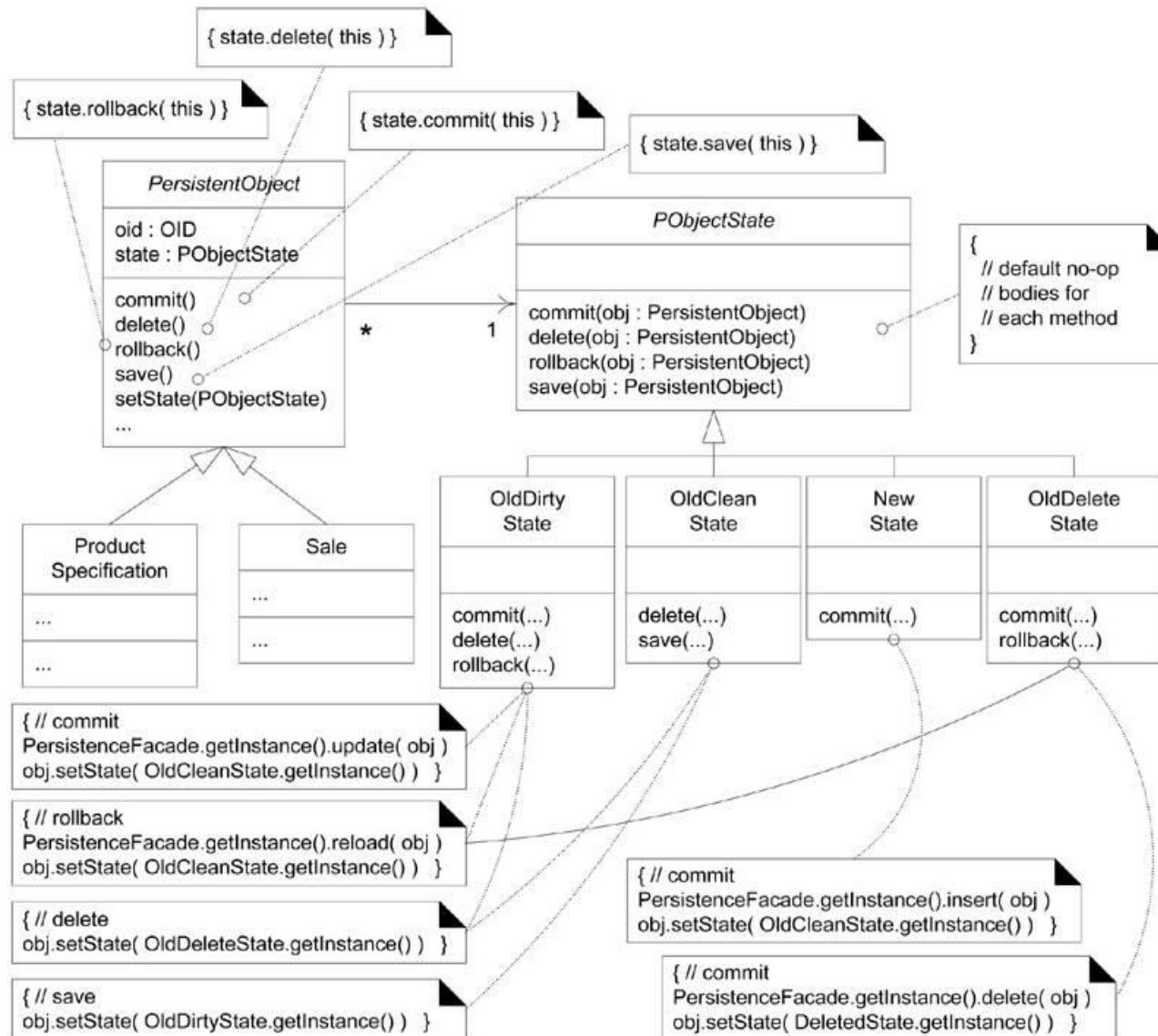
- El **comportamiento** de un objeto **cambia** dependiendo del **estado** del mismo.
- Se hace **complicado el manejo de estados** en un mismo código
- **Solución:** Se implementa una clase para cada estado diferente del objeto y el desarrollo de cada método según un estado determinado.

# State

- Contexto (Context): Mantiene una instancia de ConcreteState que referencia el estado actual
- Estado (State): Define una interfaz para el encapsulamiento de la responsabilidades asociadas con un estado particular de Context
- Estado Concreto (ConcreteState): Cada una de estas subclases implementa el comportamiento o responsabilidad de Context



# Estados Transaccionales

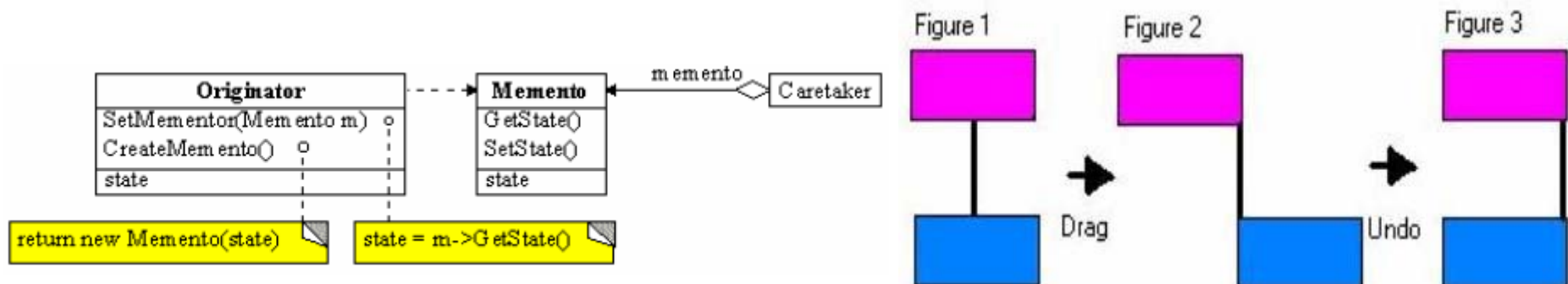


# Memento

- **Motivación:**
  - Muchas veces se quiere “retornar hacia atrás” retomando el mismo estado en ese instante  
→ Almacenar estados por cada instante.
  - **Memento** almacena parte o todo el estado interno de un objeto, para que este objeto pueda ser restaurado más tarde al estado almacenado por *Memento*.

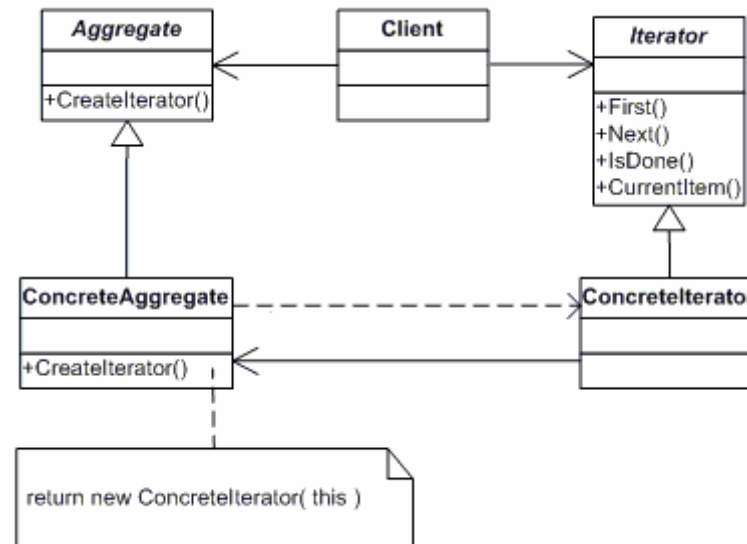
# Memento

- La composición (**Originator**) crea un objeto (**Memento**) y almacena su estado interno en él, la aplicación (**Caretaker**) mantiene una lista de los objetos (**Memento**), de tal manera que cuando el usuario realice una operación de “deshacer”, la aplicación restaure el estado de la composición (**Originator**), con lo almacenado por el objeto Memento.
- **Originator**: Crea un objeto Memento conteniendo una fotografía de su estado interno y usa a Memento para restaurar su estado interno.
- **Caretaker**: Responsable por mantener a salvo a Memento. No opera o examina el contenido de Memento.
- Puede ser usado en combinación con Command e Iterator



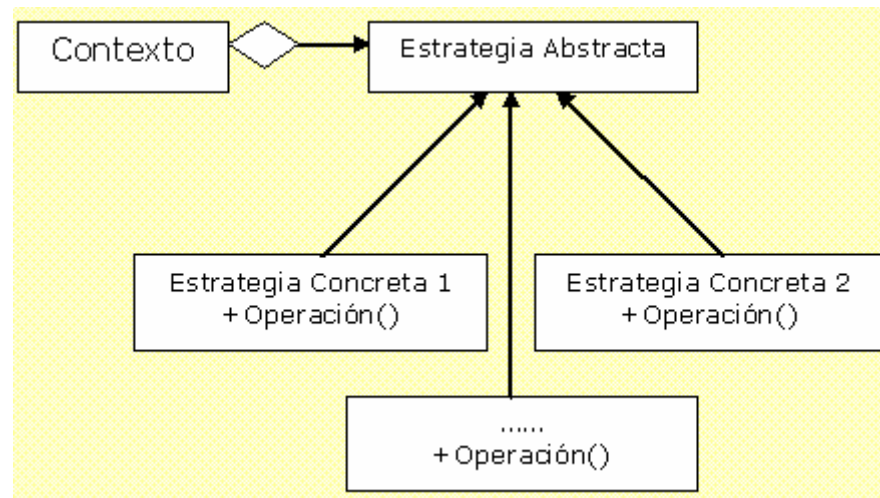
# Iterator

- Recorrido secuencial sobre los elementos de un objeto agregado sin exponer su representación subyacente.
- Abstrae el algoritmo de recorrido y protege la estructura interna del objeto agregado.
- **Solución:** Crear un puntero del tipo objeto que recorre la colección.



# Strategy

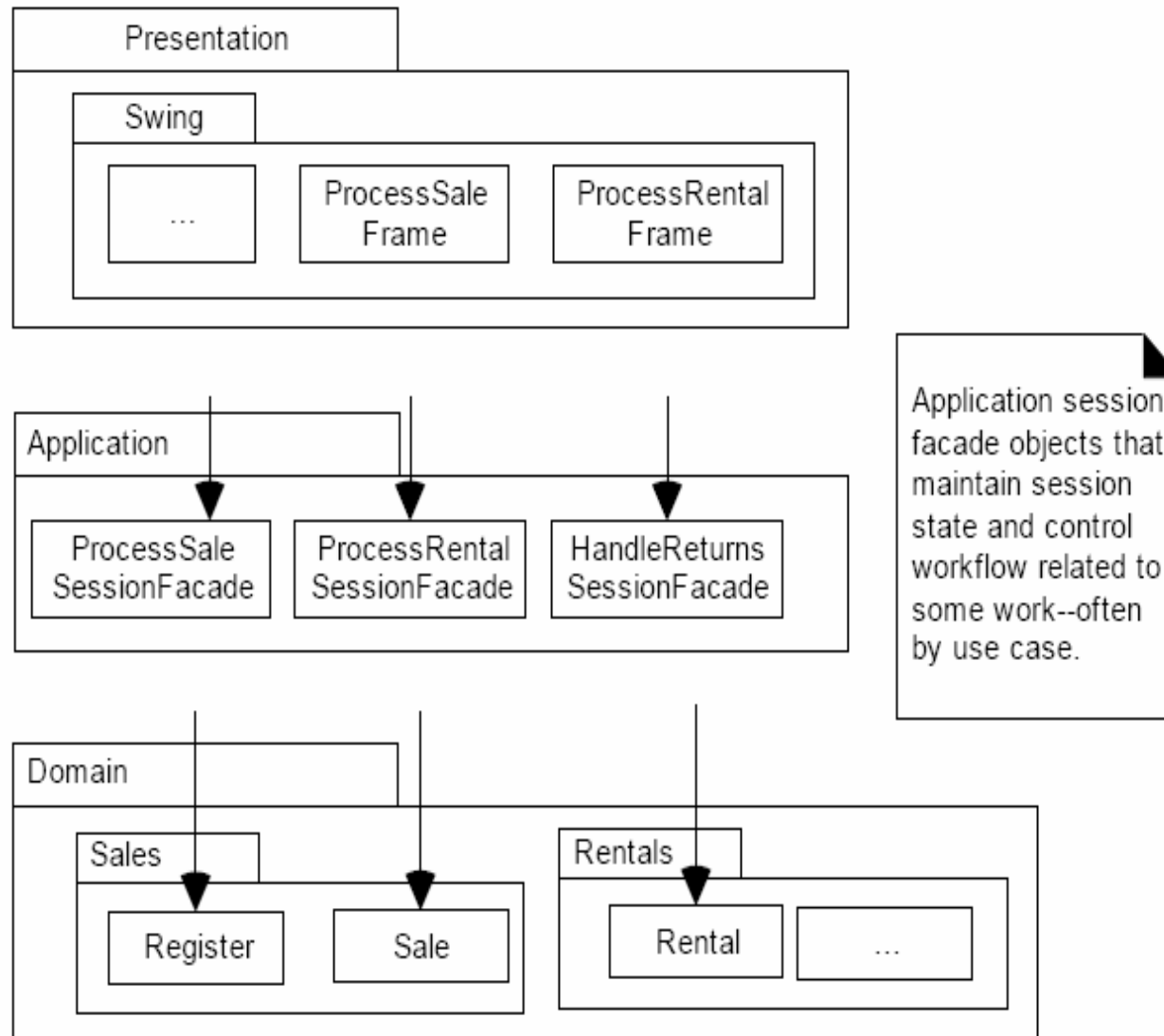
- Define una familia de algoritmos, los encapsula y los hace intercambiables.
- Los algoritmos pueden modificarse y variar independientemente del cliente que los usa.
- **Solución:** Crear una clase *Interface* con los métodos del algoritmo e implementar las distintas versiones del algoritmo en clases que implementen dicha clase *Interface*.



# Facade

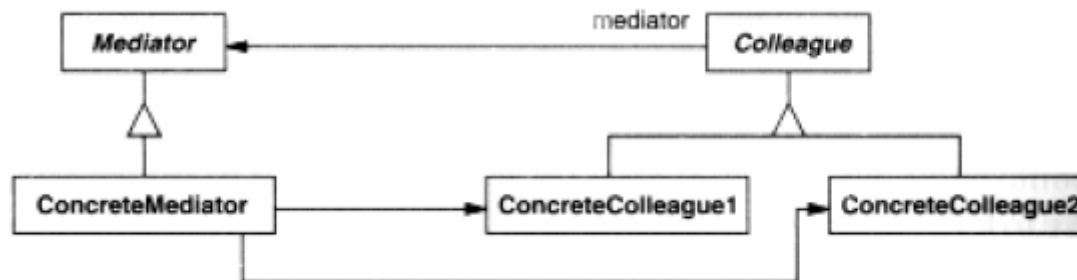
- Simplifica la interfaz entre dos sistemas o componentes de software ocultando un sistema complejo detrás de una clase que hace las veces de interfaz o fachada.

# Fachada de Sesión



# Mediator

- Define un objeto que encapsula cómo un conjunto de objetos interactúan entre sí.
- Introduce un objeto que media la comunicación entre otros objetos.
- Similar a Fachada en que abstrae la funcionalidad de clases existentes.
- Su propósito es abstraer la comunicación entre objetos colegas. Un colega debe comunicarse con el mediador en vez de comunicarse con otros colegas directamente.



# Referencias

- **Design Patterns: Elements of Reusable Object-Oriented Software. Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Addison-Wesley, 1995.**