

# Ingeniería de Software II (CI-4713)

## Patrones

mayo de 2008

# Patrones

- Solución general y repetible a un problema recurrente en diseño de software.
- Representan la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software.
- “Los patrones de diseño ayudan a aprender de los aciertos de otros en vez de los fracasos propios”. **Mark Johnson**
- **Meta**: Acelerar el proceso de desarrollo.

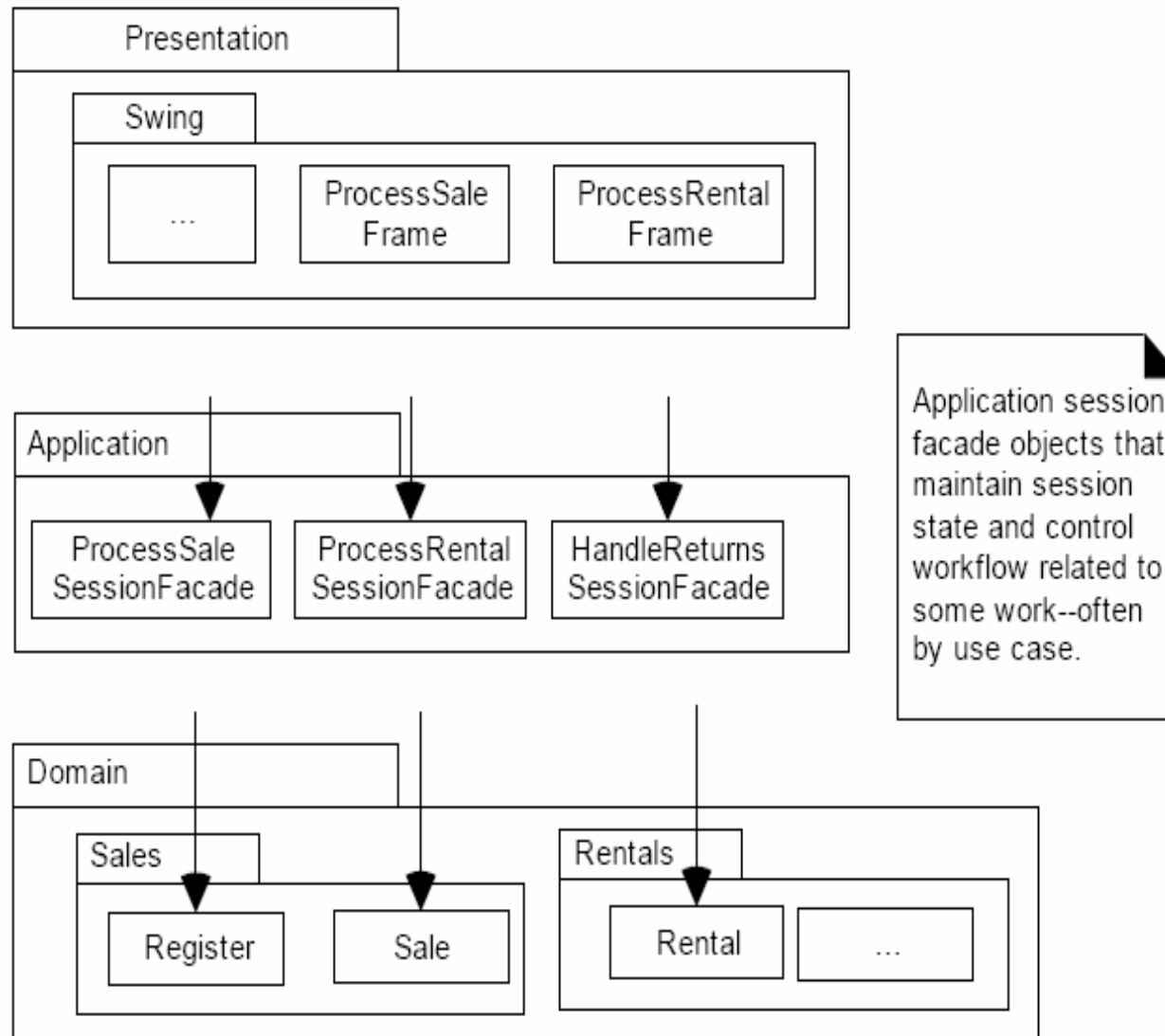
# Singleton

- **Problema:** ¿Cómo lograr disponer globalmente de una instancia de una clase y a la vez garantizar que solamente una instancia de dicha clase es creada?
- **Solución:**
  - La propia clase es responsable de crear la única instancia de ella misma.
  - El acceso global a la instancia es mediante un método de la clase.
  - Declara el constructor de la clase como privado, para que no sea instanciable directamente.

# 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.

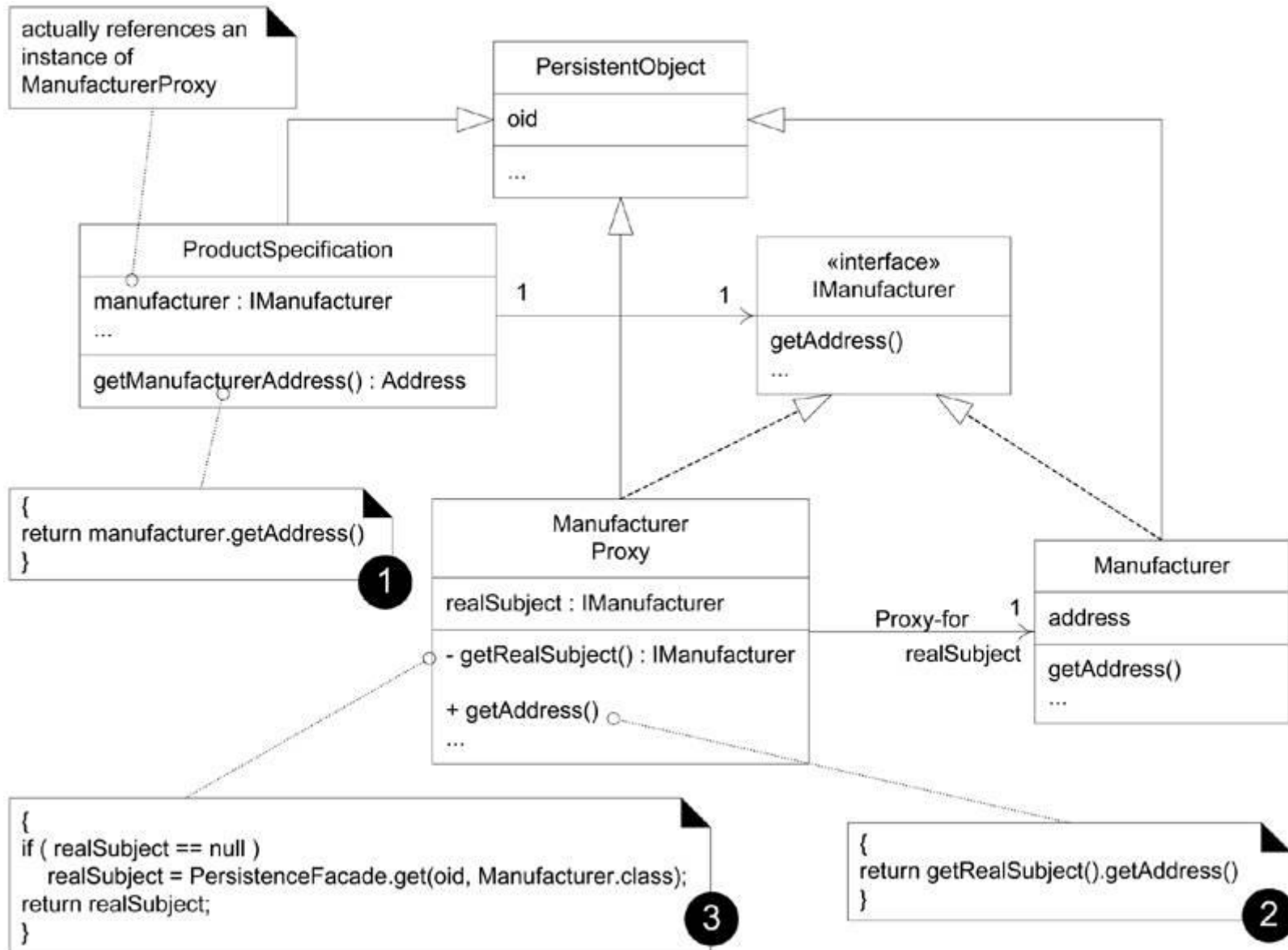
# Fachada de Sesión



# Proxy

- **Problema:**
  - Lentitud de carga.
  - Costo de creación de objetos de poca demanda.
  - Objetos ubicados en diferentes espacios físicos.
- **Propósito**
  - Un objeto actúa como sustituto de otro y permite controlar el acceso a dicho objeto.

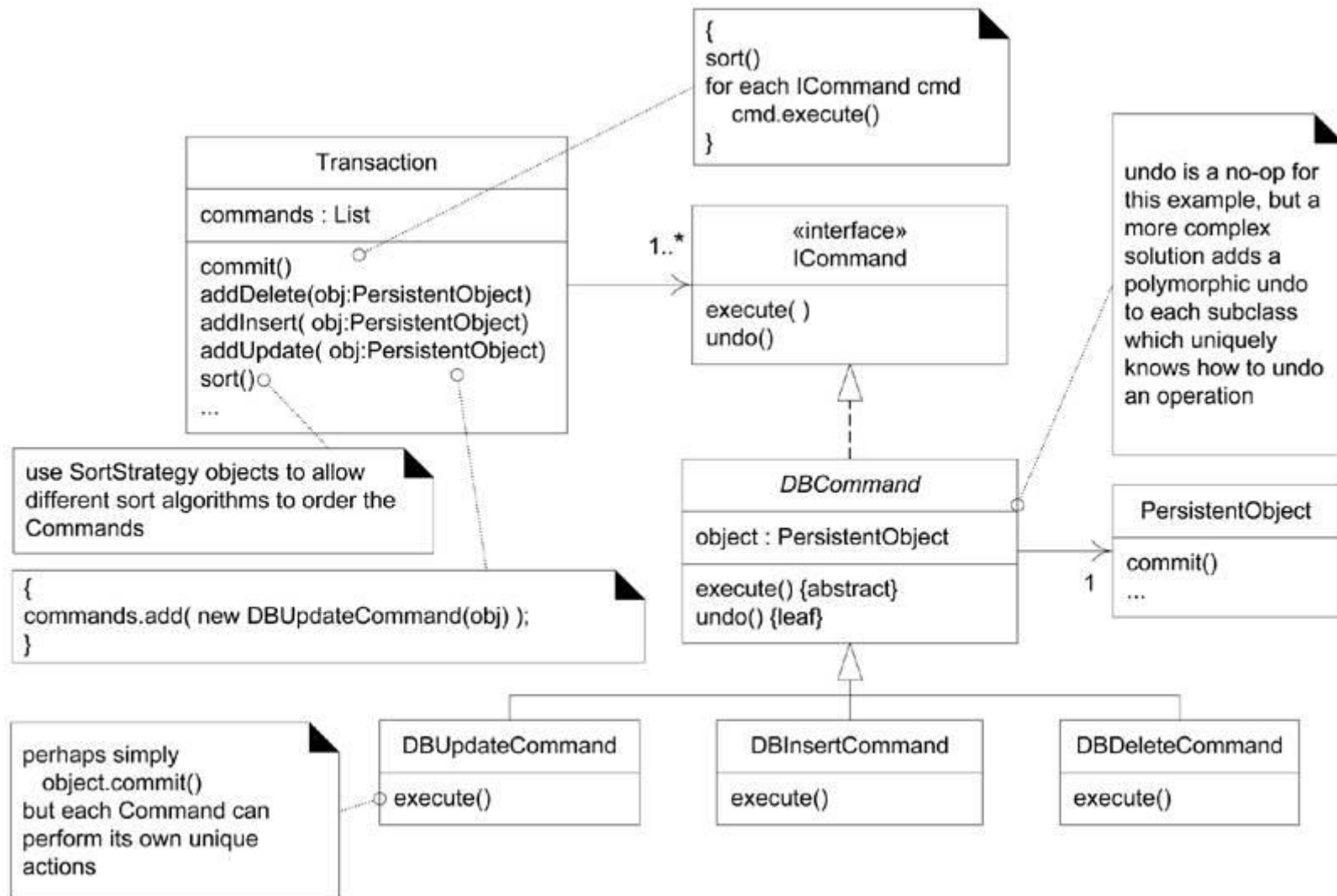
# Materialización Perezosa



# Command

- Solicita una operación a un objeto sin conocer realmente el contenido de la operación.
- Encapsula la operación como objeto.

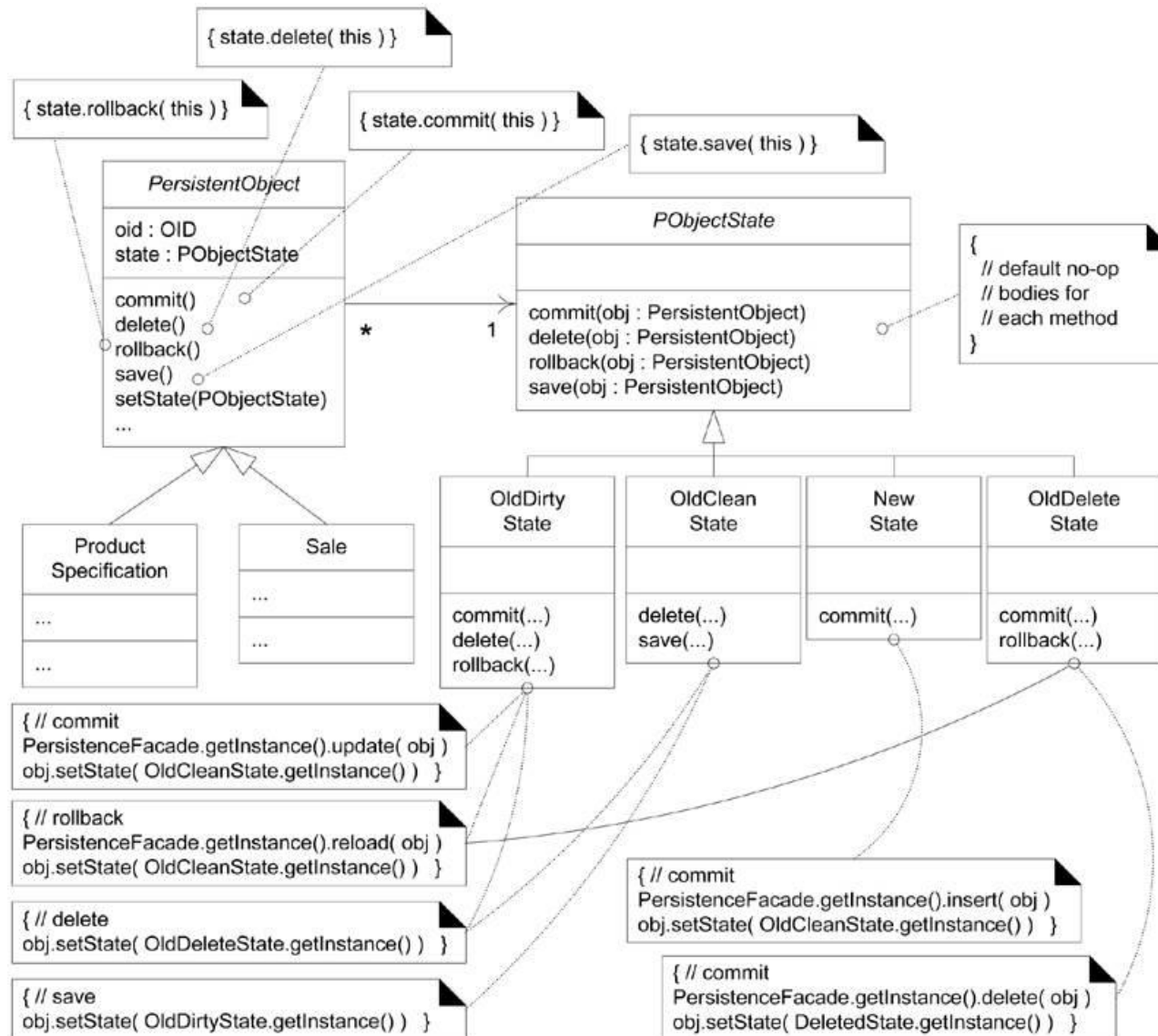
# Diseño de una transacción con el Patrón Comando



# State

- El comportamiento de un objeto cambia dependiendo del estado del mismo.
- Solución:
  - Se implementa una clase para cada estado diferente del objeto y el desarrollo de cada método según un estado determinado.

# Estados Transaccionales



# Template Method

- **Problema**
  - Gran cantidad de algoritmos con una estructura o un comportamiento similar.
- **Propósito**
  - Define un esqueleto de un algoritmo, dejando los detalles a las subclases. Permite redefinir ciertos pedazos del algoritmo sin cambiar su estructura.

# Materialización con el Patrón Plantilla

- **Redefinición del método de Enganche (Hook)**

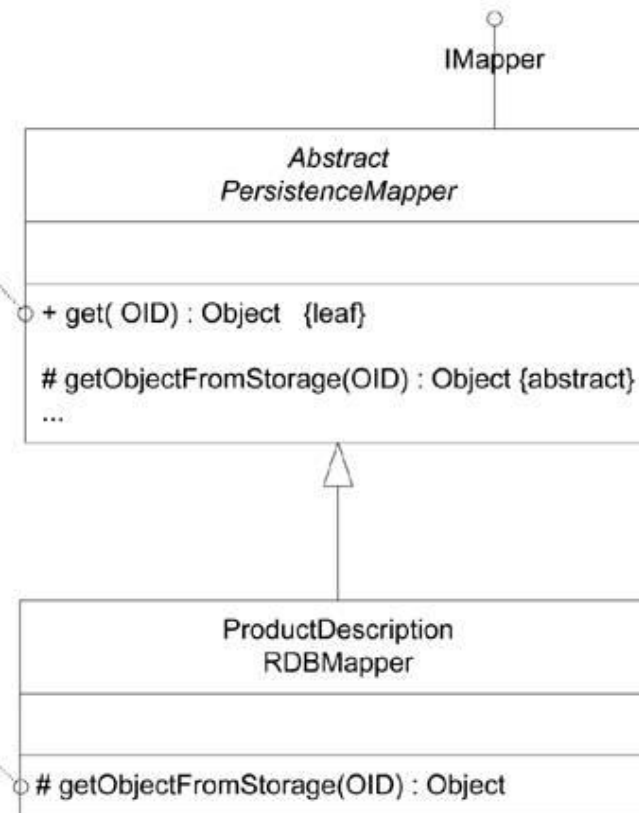
```
// template method
public final Object get( OID oid )
{
    obj := cachedObjects.get(oid);
    if (obj == null )
    {
        // hook method
        obj = getObjectFromStorage( oid );

        cachedObjects.put( oid, obj )
    }
    return obj
}
```

```
// hook method override
protected Object getObjectFromStorage( OID oid )
{
    String key = oid.toString();
    dbRec = SQL execution result of:
        "Select * from PROD_DESC where key =" + key

    ProductDescription pd = new ProductDescription();
    pd.setOID( oid );
    pd.setPrice( dbRec.getColumn("PRICE" ) );
    pd.setItemID( dbRec.getColumn("ITEM_ID" ) );
    pd.setDescrip( dbRec.getColumn("DESC" ) );

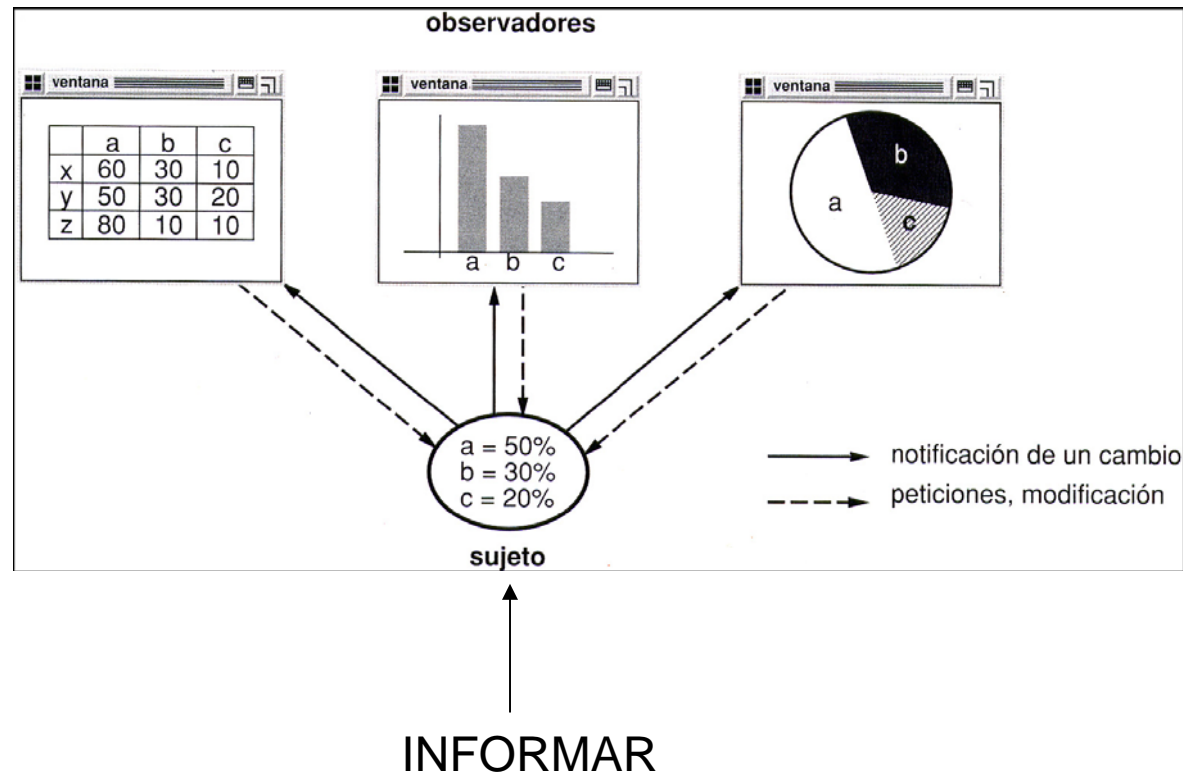
    return pd;
}
```



# Observador

- Define una dependencia uno a muchos entre objetos de modo que cuando el estado de un objeto cambia, se les notifica el cambio a todos los que dependen de él y se actualizan de forma automática.
- ¿Qué hacer cuando las capas más bajas necesitan comunicarse con las capas de arriba?
  - Usar Patrón *Observador*.

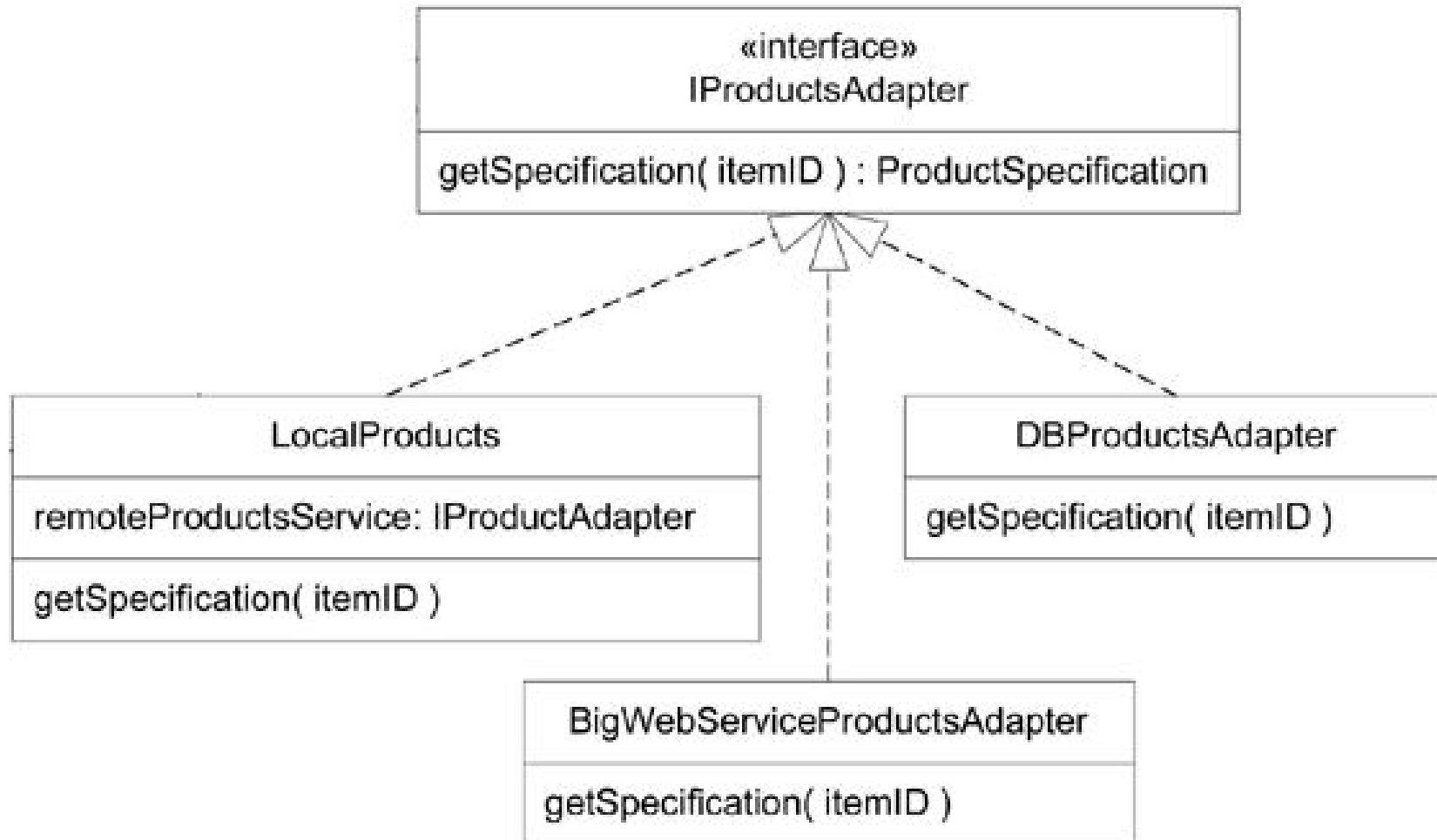
# Observador



# Adapter

- Se utiliza para transformar una interfaz en otra, de tal modo que una clase que no pudiera utilizar la primera, haga uso de ella a través de la segunda.

# Adapter



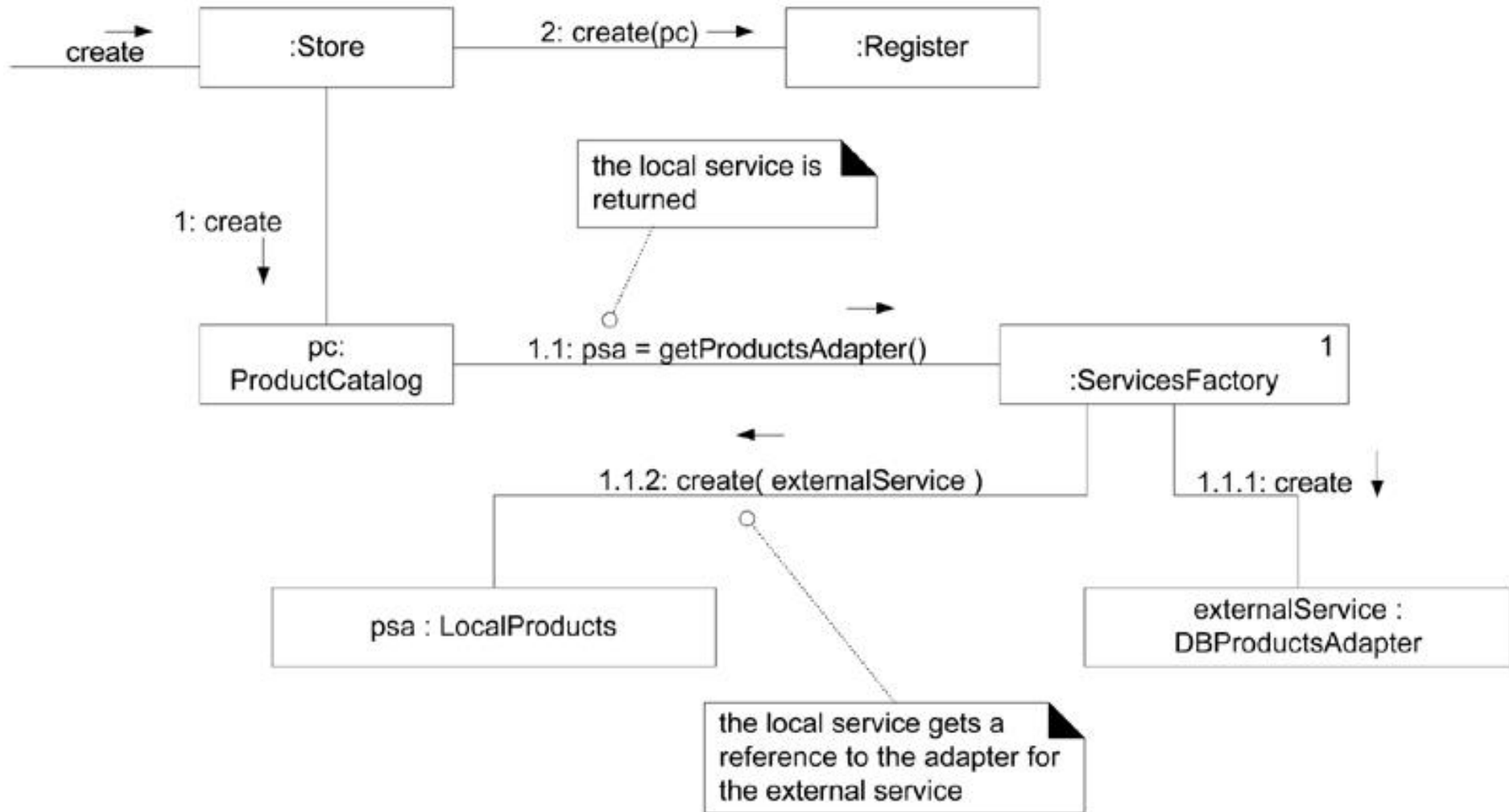
# Abstract Factory

- Encapsula un grupo de objetos que tienen un tema en común.
- Una *Fábrica* es la ubicación en el código donde los objetos son construidos.
- Nuevos tipos derivados son incluidos sin afectar el código que usa el objeto base.

# Factory

- Problema:
  - ¿Quién crea el adaptador?
  - ¿Cómo determinar qué clase de adaptador crear?
- Solución:
  - Crear un objeto de fabricación pura denominado **Factory** que maneje la creación.

# Inicialización de Servicios



# Referencias

- **UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado. Larman, C. 2da Edición. Addison Wesley Professional. 2002.**