

Ingeniería de Software II (CI-4713)

Persistencia

mayo de 2008

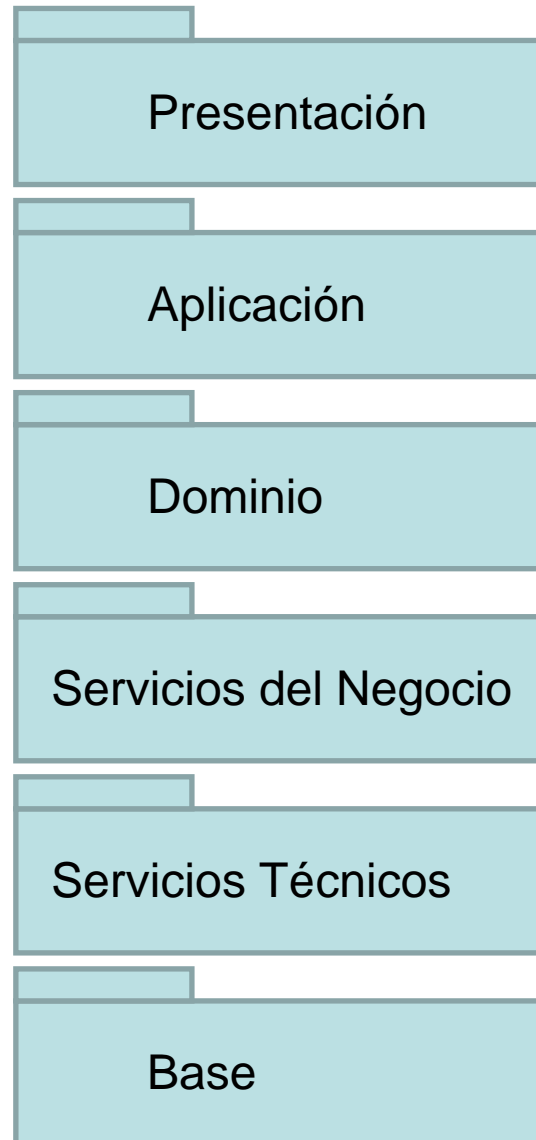
Mecanismos de Almacenamiento

- **BDO:**
 - No se necesitan servicios de persistencia.
- **BDR:**
 - Se requiere un servicio de correspondencia entre objetos y relaciones.
- **XML, archivos, BD jerárquicas, etc:**
 - Incompatibilidad entre estos formatos y las representaciones de los objetos.

Solución

- Un **Framework de Persistencia**: Conjunto de tipos de propósito general, reutilizable y extensible, que proporciona funcionalidad para dar soporte a los objetos persistentes.
- **Servicio de Persistencia**: Traduce los objetos a registros, los almacena en una BD y traduce los registros a objetos.

Arquitectura



Un **Servicio de Persistencia** es un subsistema dentro de la capa de **Servicios Técnicos**

Framework

- Conjunto cohesivo de **interfaces** y **clases** que colaboran para proporcionar los servicios de un sistema lógico.
- Contiene **clases concretas** y **abstractas**.
- El usuario **puede definir subclases** de las clases existentes del *framework*.
- Son **reutilizables**.

Framework de Persistencia

- **Requisitos**

- Almacenar y recuperar los objetos en un mecanismo de almacenamiento persistente.
- Confirmar y deshacer las transacciones
- Diseño extensible para dar soporte a diferentes mecanismos de almacenamiento: BDR, archivos, documentos XML.

- **Aspectos importantes:**

- Correspondencia de esquemas

Framework de Persistencia

- **Aspectos importantes:**
 - Identidad de objetos y registros
 - Materialización (transformación de una representación no O.O. en objetos) y desmaterialización (opuesto a la materialización).
 - Conversor de BD: Responsable de la materialización y desmaterialización.
 - Almacenar los objetos materializados en caché

Framework de Persistencia

- **Aspectos importantes:**
 - Estado de la transacción de objetos para determinar si es necesario almacenar nuevamente los objetos persistentemente.
 - Operaciones de transacción: Commit y Rollback.
 - Materialización perezosa: Una instancia es materializada bajo demanda.
 - *Proxies* virtuales para implementar la materialización perezosa.

Correspondencia

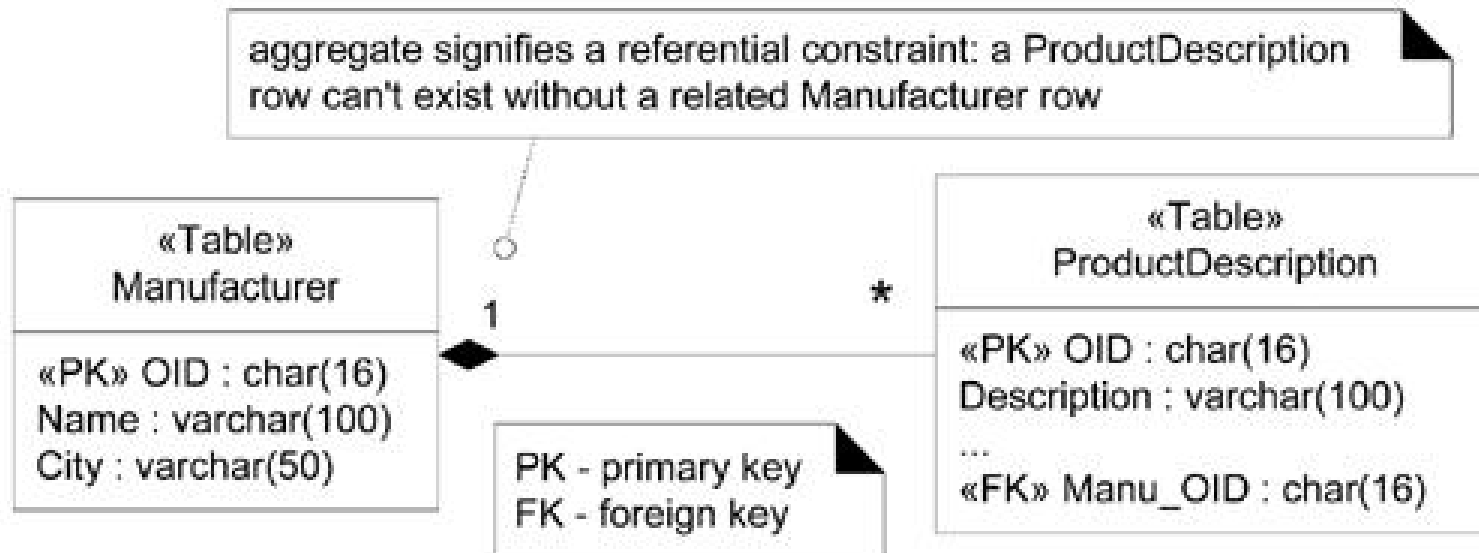
- Uso del Patrón Representación de Objetos como Tablas
 - Definir una tabla en una BDR por cada clase de objeto persistente.
 - Los atributos cuyos tipos de datos son primitivos (número, cadena de texto, booleano, etc) se corresponden con las columnas. (1FN)
 - Relaciones 1:1:
 - Crear una clave foránea en una o ambas tablas.
 - Crear una tabla asociación con ambos OIDs

Correspondencia

- Uso del Patrón Representación de las Relaciones de los Objetos en Tablas
 - Relaciones 1:N:
 - **Crear una clave foránea (del lado N).**
 - Crear una tabla asociación con ambos OIDs
 - Relaciones M:N
 - Crear una tabla asociación con ambos OIDs
- ¿Qué sucede con Superclases / SubClases, Agregaciones, Clases Asociaciones, cardinalidades dadas?

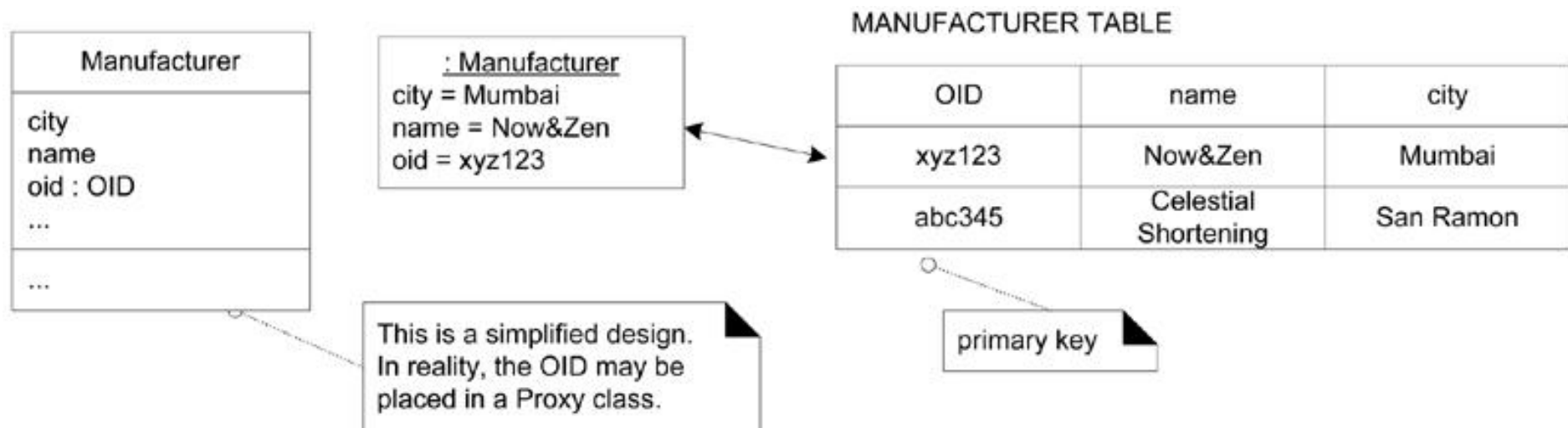
Perfil (Profile) de Modelado de Datos

- Un perfil es un conjunto coherente de estereotipos de UML, valores etiquetados y restricciones para un propósito específico.



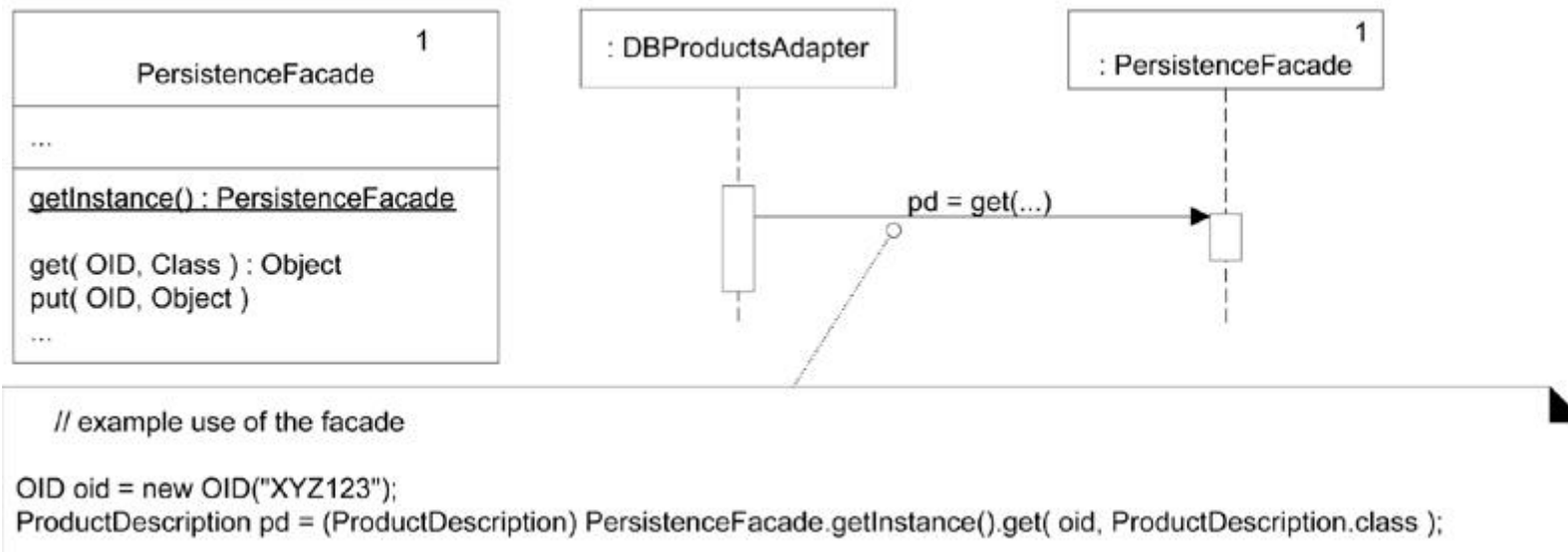
Identidad de Objeto

- Asignar un identificador de objeto (OID) a cada registro y objeto.



Acceso al Servicio de Persistencia

- Definir una fachada
- Operaciones:
 - Recuperar un objeto dado su OID
 - Conocer el tipo de Objeto que se va a materializar



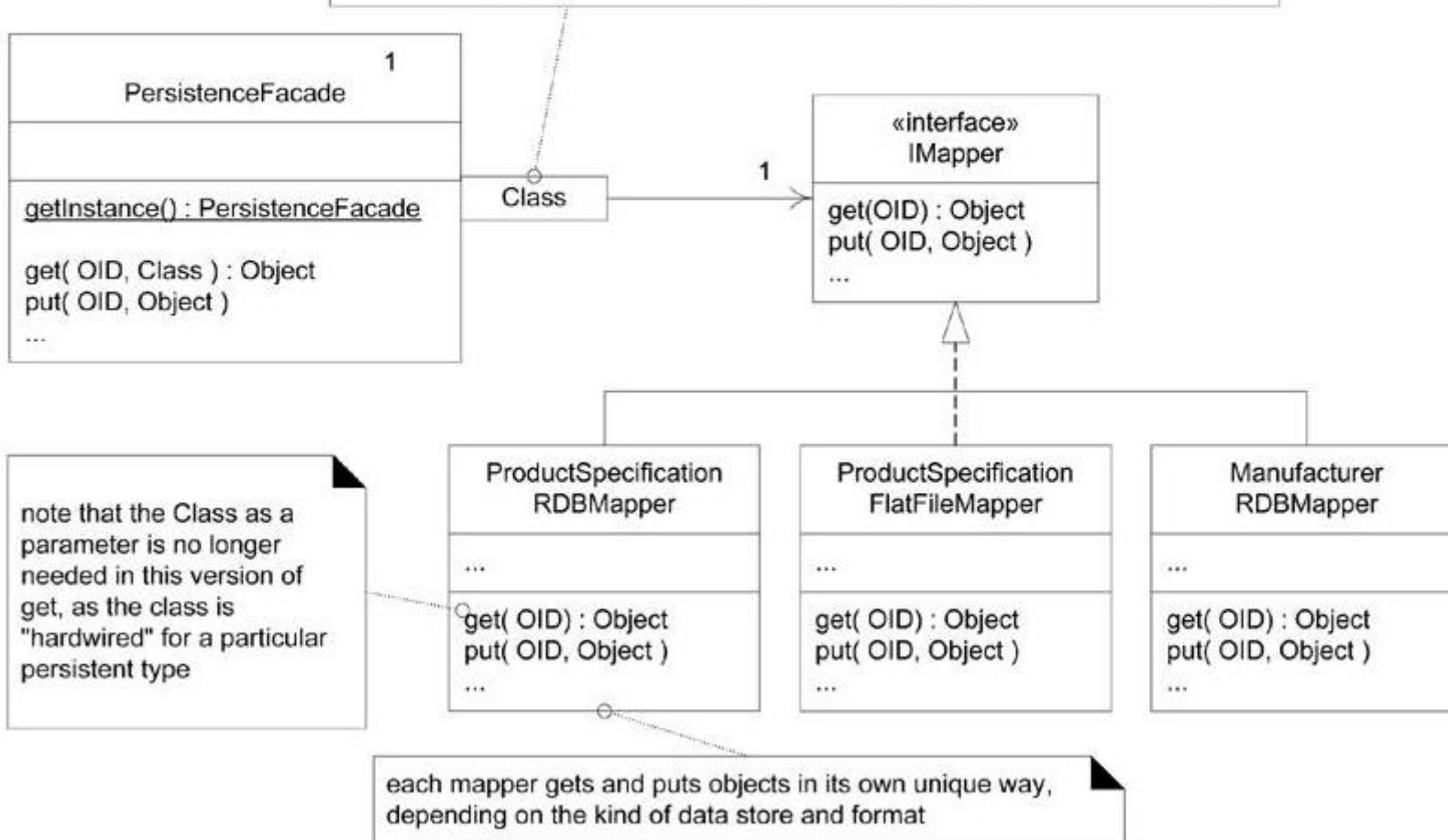
Conversor de BD

- Usar el patrón Intermediario (Broker) o Conversor (Mapper) de BD
- Proponer crear una clase que sea responsable de materializar y desmaterializar un objeto almacenado.

Conversor de BD

UML notation: This is a qualified association. It means:

1. There is a 1-M association from PersistenceFacade to IMapper objects.
2. With a key of type Class, an IMapper is found (e.g., via a HashMap lookup)



Conversor de BD

```
class PersistenceFacade
{
//...

    public Object get( OID oid, Class persistenceClass )
    {
        // an IMapper is keyed by the Class of the persistent
        object IMapper mapper =
            (IMapper) mappers.get( persistenceClass );
        // delegate
        return mapper.get( oid );
    }
//...
}
```

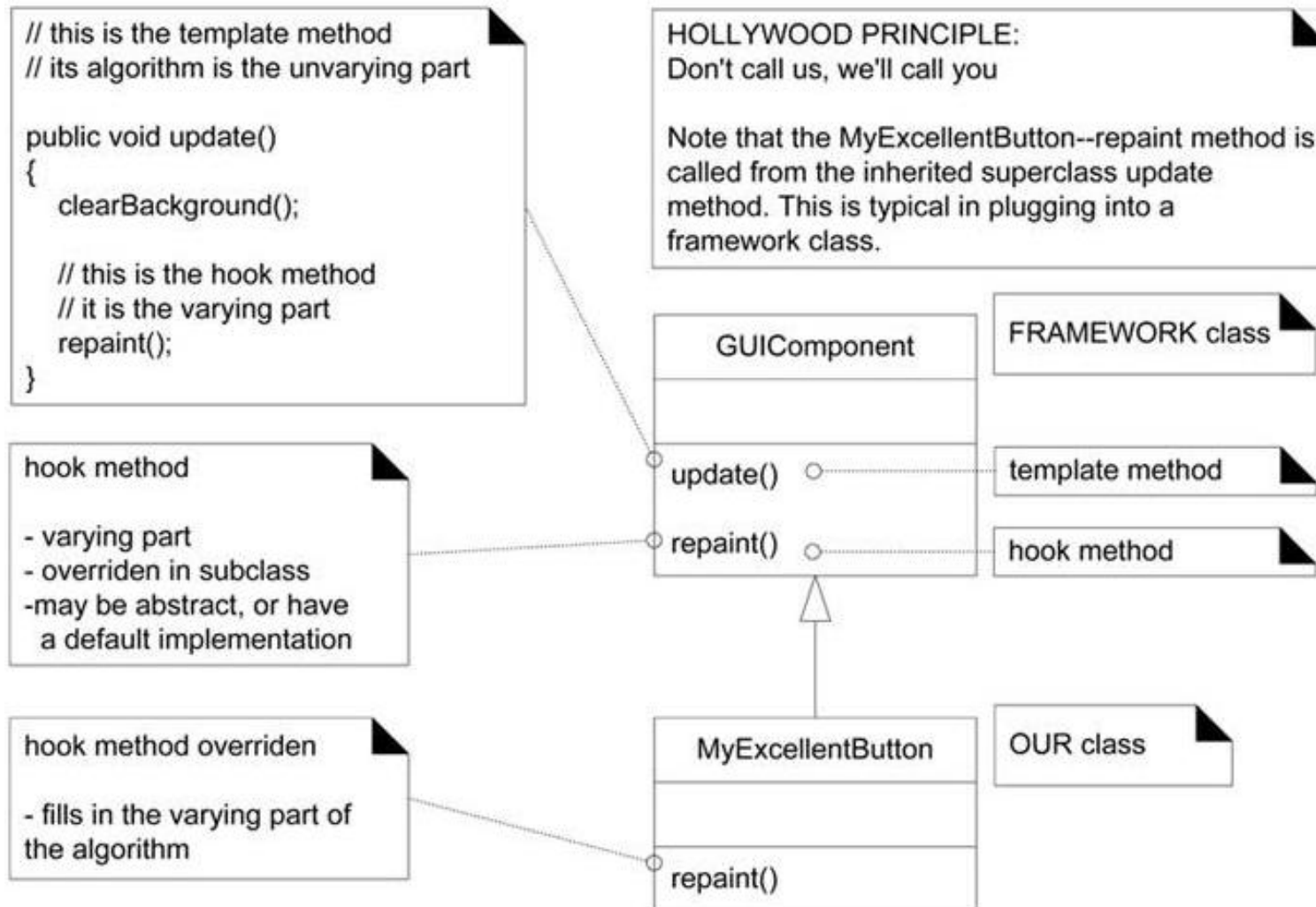
Conversores basados en Metadatos

- Generan dinámicamente la correspondencia entre un esquema de objetos y otro esquema (relacional por ejemplo) en base a la lectura de metadatos que describen la correspondencia.

Diseño del Framework con el Patrón Plantilla

- Crear un método (Método Plantilla) en una superclase que define el esqueleto de un algoritmo, con sus partes variables e invariables.
- El método plantilla invoca otros métodos.
- Las subclases pueden redefinir los métodos que varían.

Diseño del Framework con el Patrón Plantilla



Materialización con el Patrón Plantilla

if (object in cache)

 return it

else

 create the object from its representation in storage

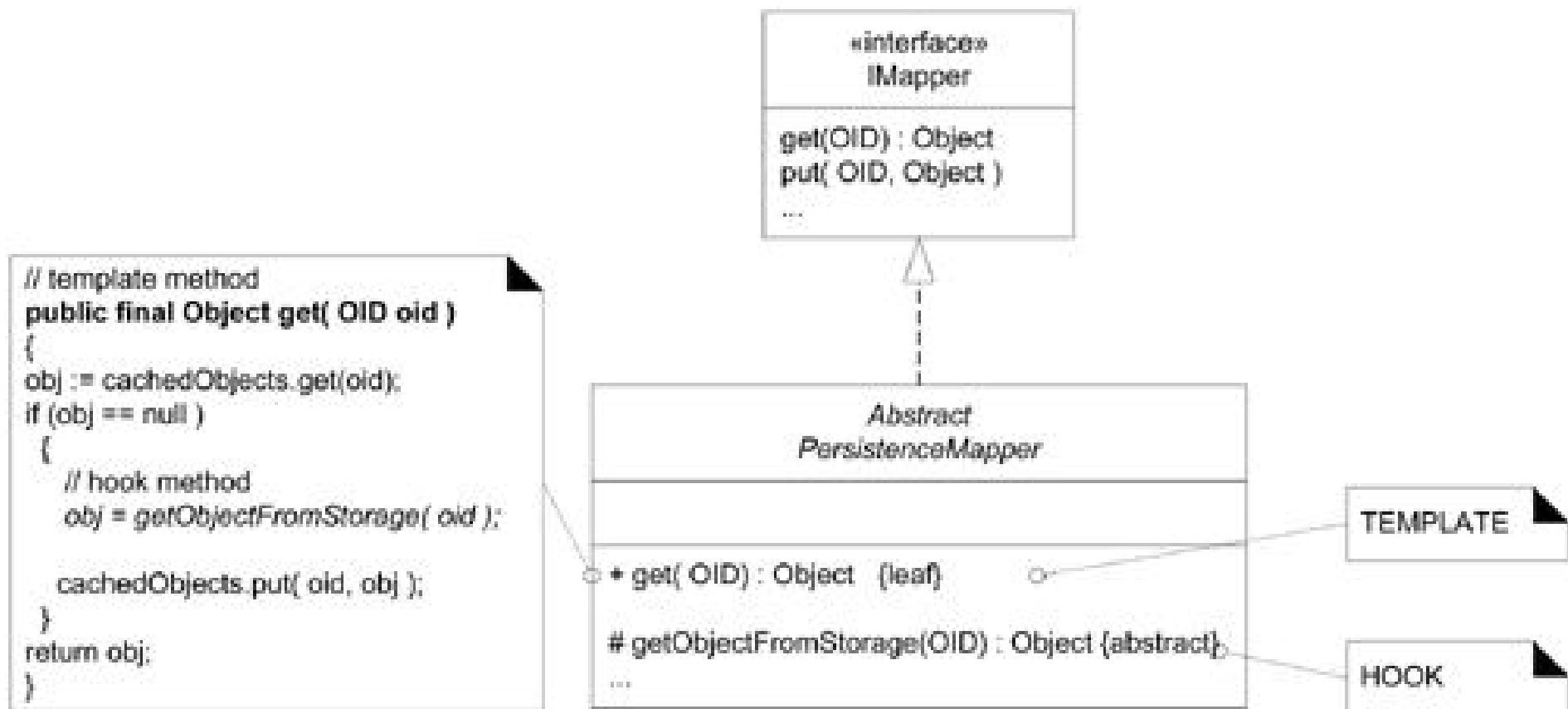
 save object in cache

 return it

- **Variación:** La manera de crear el objeto a partir de su almacenamiento.

Materialización con el Patrón Plantilla

- **Método Plantilla para Objetos Conversores**



Materialización con el Patrón Plantilla

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

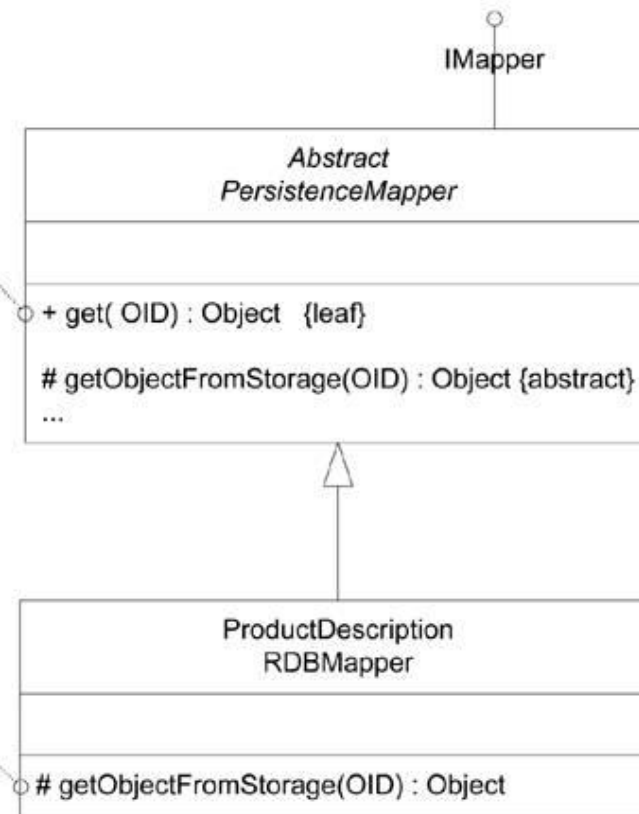
```
// template method
public final Object get( OID oid )
{
  obj := cachedObjects.get(oid);
  if (obj == null )
  {
    // hook method
    obj = getObjectFromStorage( oid );

    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

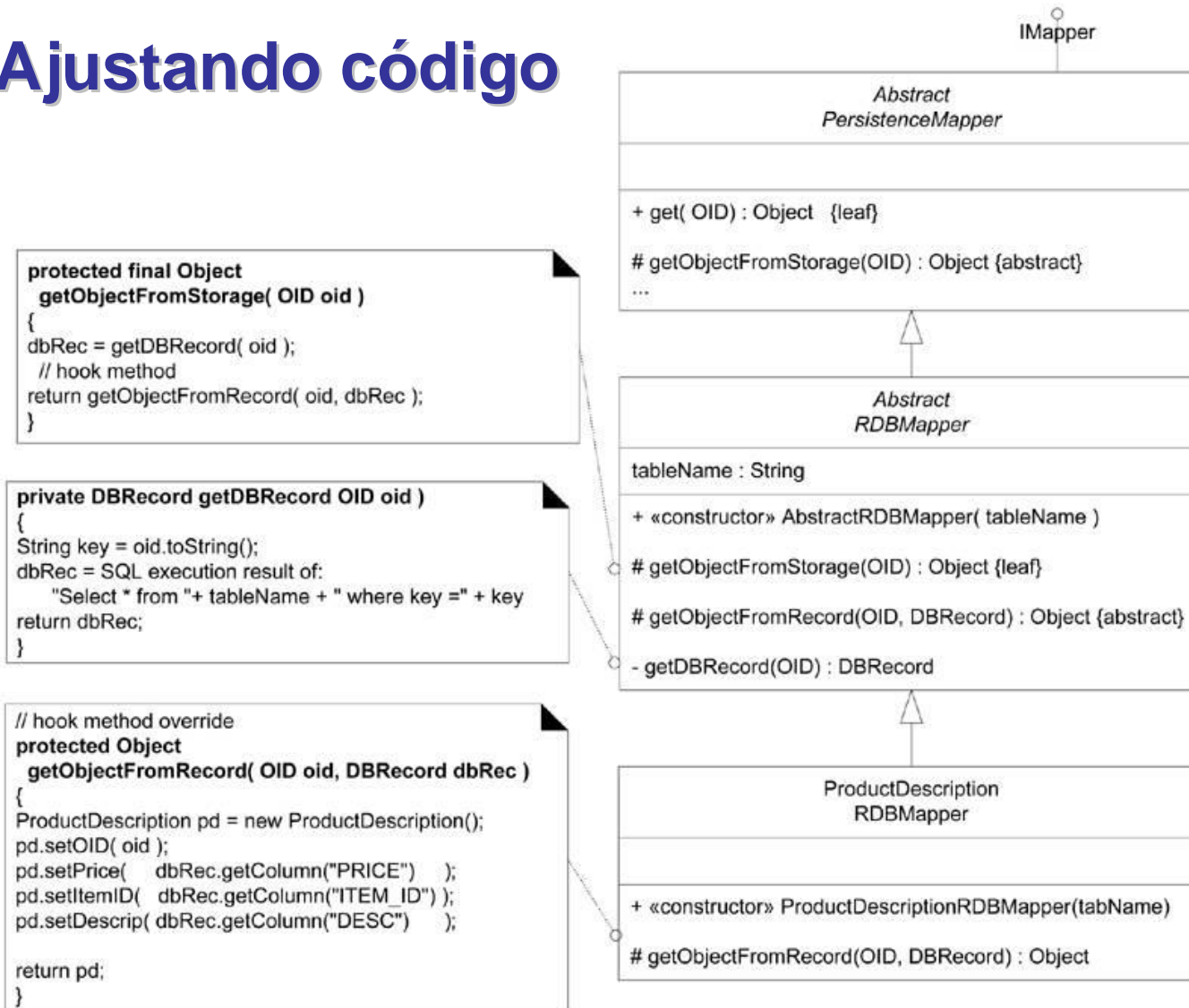
  ProductDescription pd = new ProductDescription();
  pd.setOID( oid );
  pd.setPrice( dbRec.getColumn("PRICE" ) );
  pd.setItemID( dbRec.getColumn("ITEM_ID" ) );
  pd.setDescrip( dbRec.getColumn("DESC" ) );

  return pd;
}
```

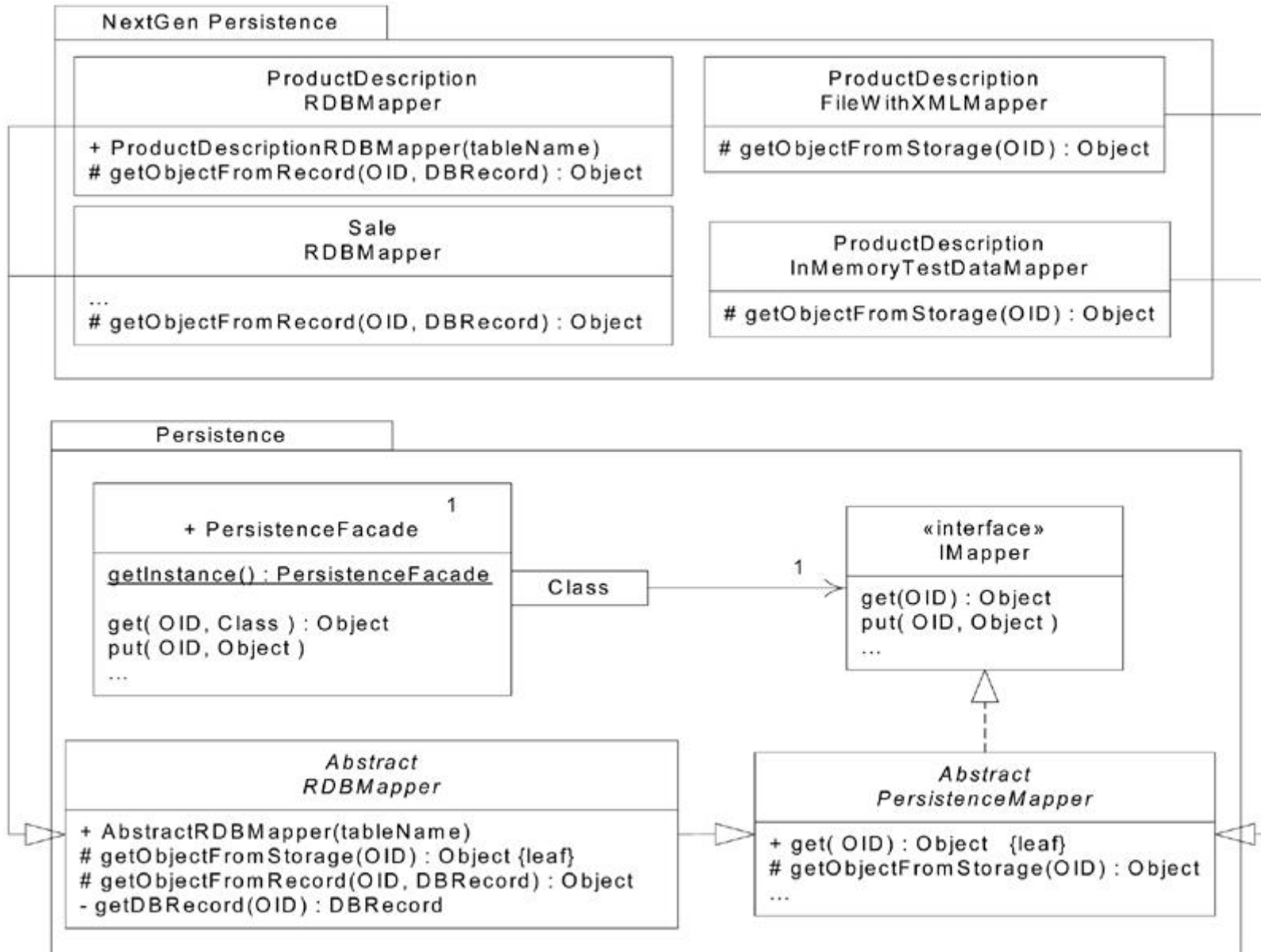


Materialización con el Patrón Plantilla

- **Ajustando código**



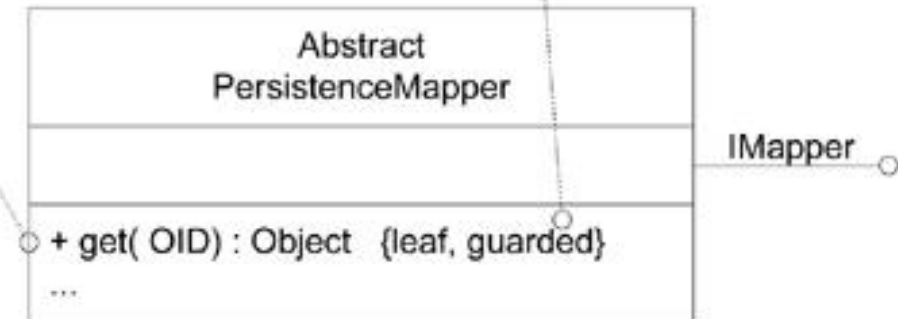
Framework de Persistencia



Métodos Sincronizados

```
// Java  
public final synchronized Object get( OID oid )  
{ ... }
```

{guarded} means a "synchronized" method; that is, only 1 thread may execute at a time within the family of guarded methods of this object.



Configuración de conversores

1.

```
class MapperFactory {  
    public IMapper getProductDescriptionMapper() {...}  
    public IMapper getSaleMapper() {...}  
    ...  
}
```
2.

```
class MapperFactory {  
    public Map getAllMappers() {...} ... }
```

 - ```
class PersistenceFacade {
 private java.util.Map mappers =
 MapperFactory.getInstance().getAllMappers(); ...}
```

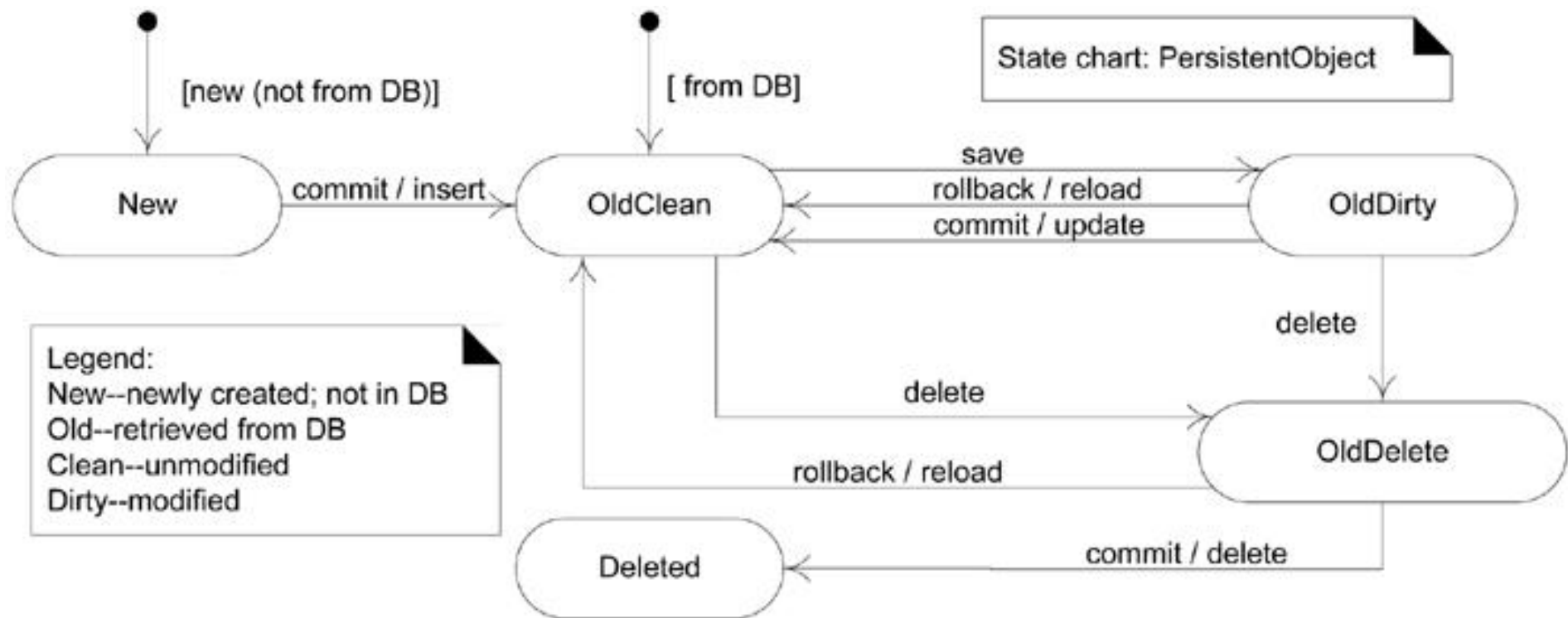
# Gestión de Caché

- El patrón Gestión de Caché propone que el conversor de BD sea el responsable de mantener la caché.
- El conversor de BD busca primero en caché evitando materializaciones innecesarias.

# Reunir y Ocultar SQLs en una clase

- class RDBOperations {  
public ResultSet getProductDescriptionData( OID oid ) {...}  
public ResultSet getSaleData( OID oid ) {...} ... }
- class ProductDescriptionRDBMapper extends  
AbstractPersistenceMapper {  
protected Object getObjectFromStorage( OID oid ) {  
    ResultSet rs =  
    RDBOperations.getInstance().getProductDescriptionData( oid );  
    ProductDescription ps = new ProductDescription();  
    ps.setPrice( rs.getDouble( "PRICE" ) );  
    ps.setOID( oid );  
    return ps; }  
}

# Estados Transaccionales

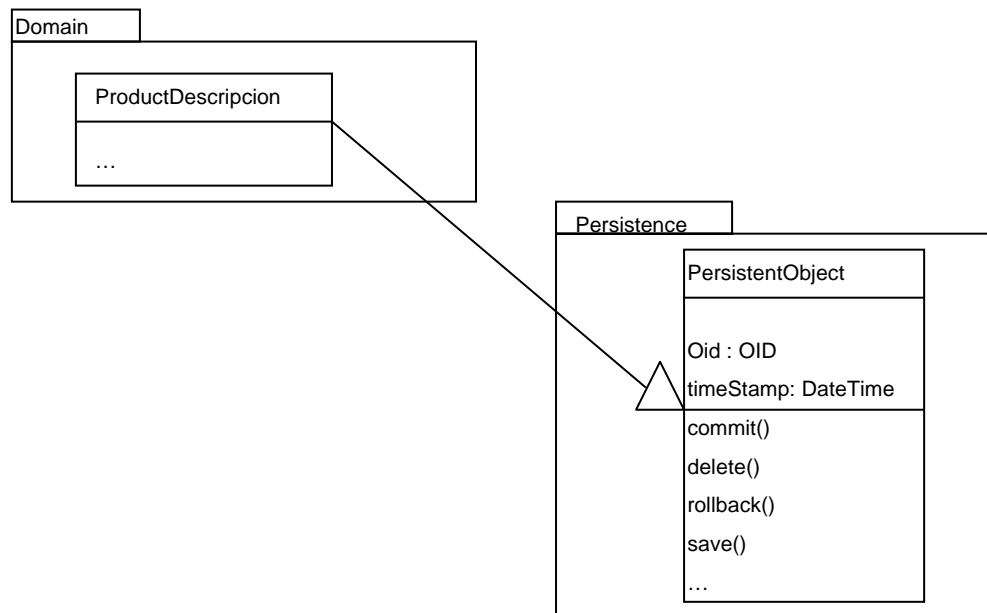


# Patrón Estado

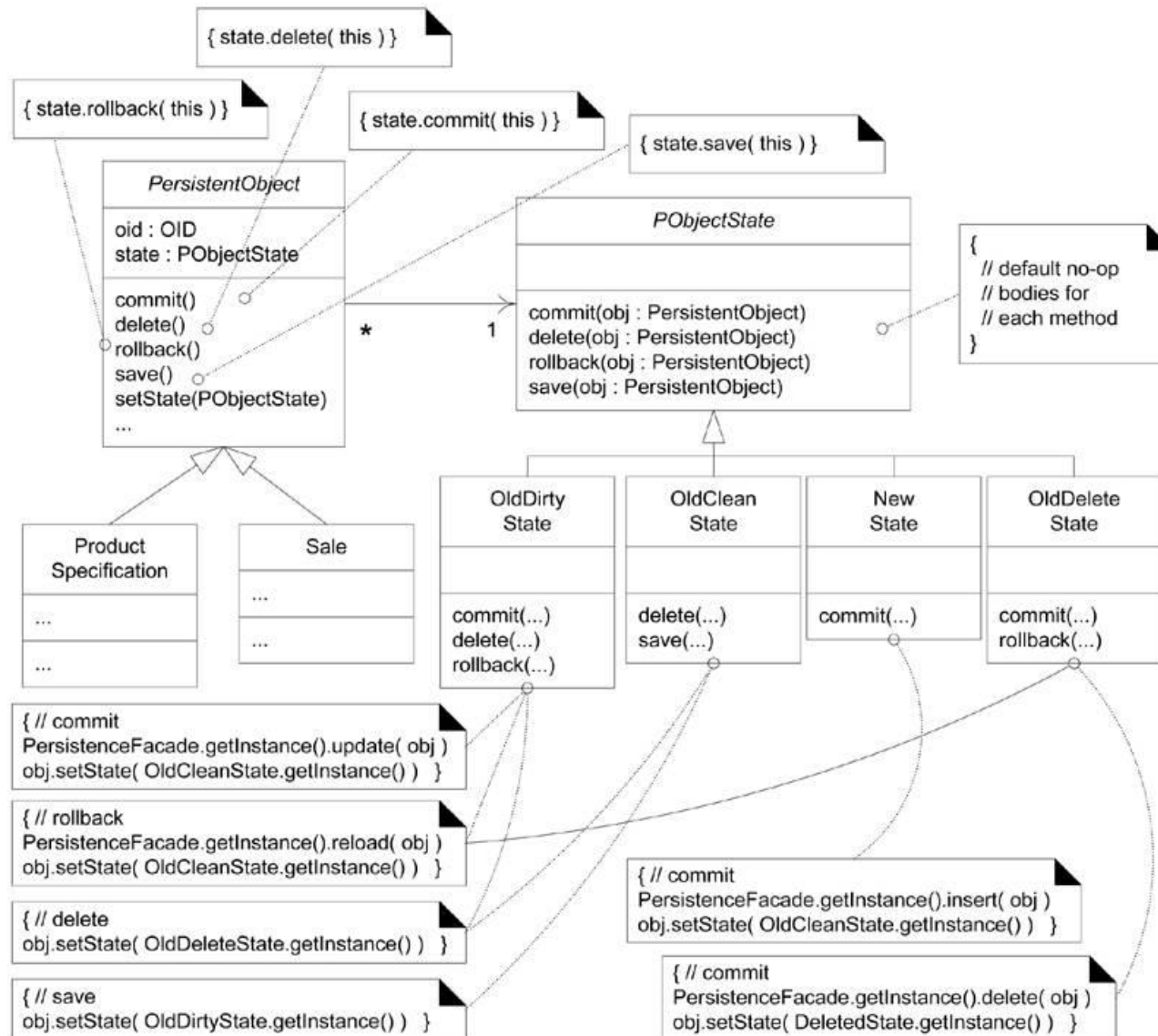
- Problema:
  - El comportamiento de un objeto depende de su estado y sus métodos contienen la lógica de casos que reflejan las acciones condicionales dependiente del estado.
- Solución:
  - Cree clases estado para cada estado, que implementan una interfaz común.

# Estados Transaccionales

- Si toda clase del dominio es subclase del Objeto Persistente



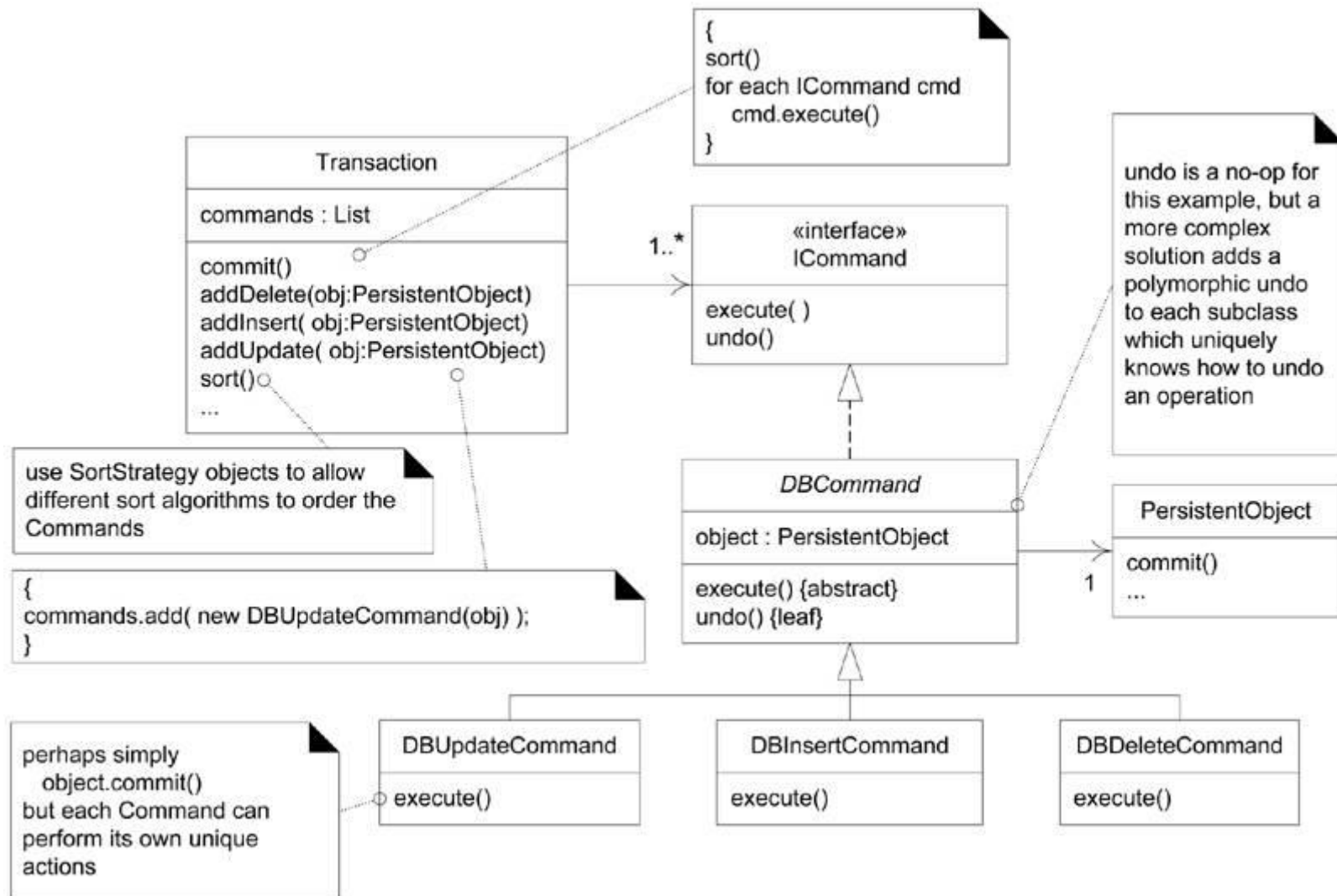
# Estados Transaccionales



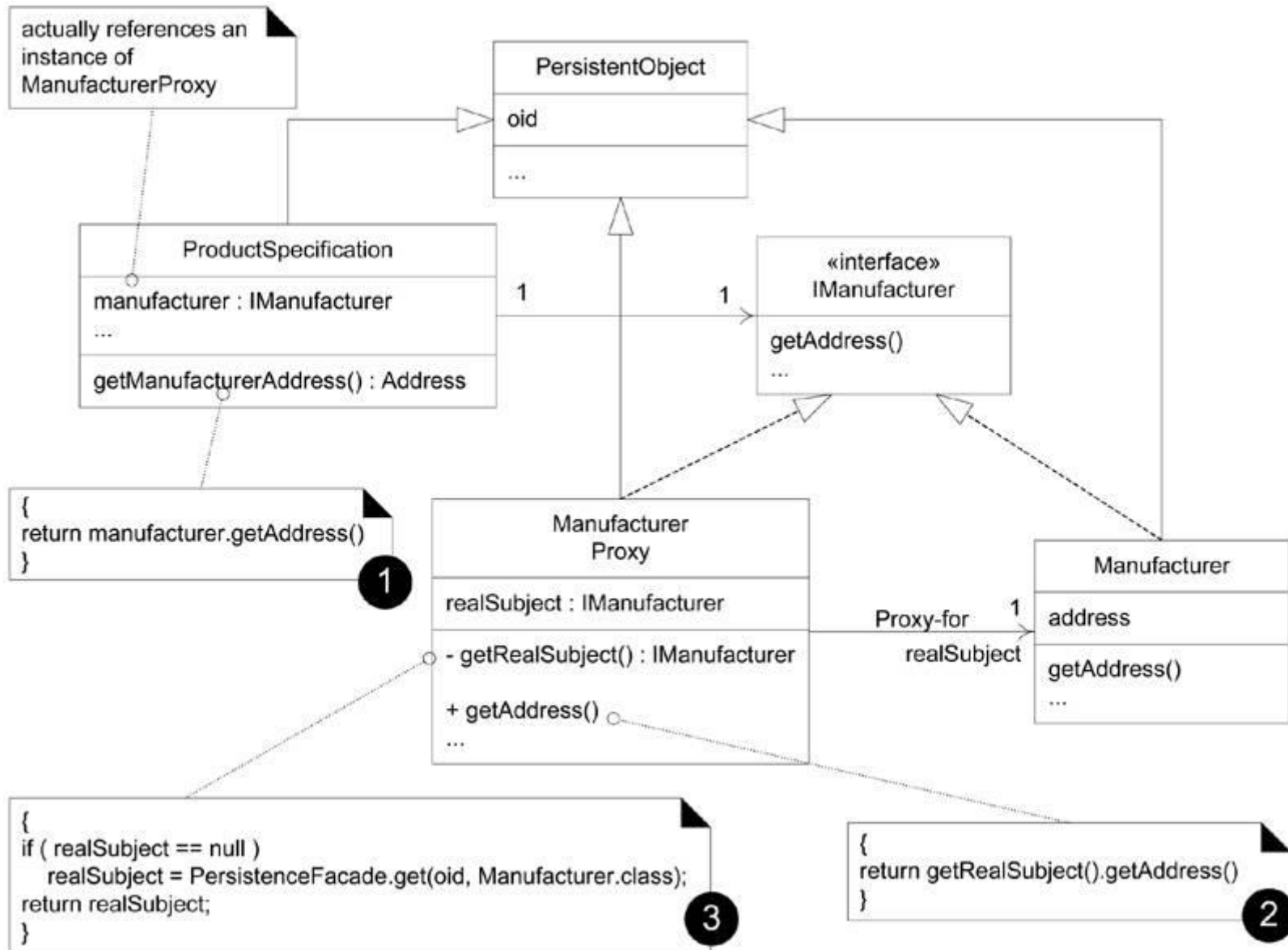
# Diseño de una transacción con el Patrón Comando

- Problema:
  - ¿Cómo gestionar las solicitudes o tareas que necesitan funciones como ordenar (estableciendo prioridades), poner en cola, retrasar, anotar en registro o deshacer?.
- Solución:
  - Defina una clase por cada tarea que implemente una interfaz común.

# Diseño de una transacción con el Patrón Comando



# Materialización Perezosa



# Materialización Temprana de Fabricante

```
class ProductSpecificationRDBMapper extends AbstractPersistenceMapper {
 protected Object getObjectFromStorage(OID oid) {
 ResultSet rs =
 RDBOperations.getInstance().getProductSpecificationData(oid);
 ProductSpecification ps = new ProductSpecification();
 ps.setPrice(rs.getDouble("PRICE"));
 // here's the essence of it
 String manufacturerForeignKey = rs.getString("MANU_OID");
 OID manuOID = new OID(manufacturerForeignKey);
 ps.setManufacturer((Manufacturer)
 PersistenceFacade.getInstance().get(manuOID, Manufacturer.class));
 }
}
```

# Materialización Temprana de Fabricante

```
class ProductSpecificationRDBMapper extends AbstractPersistenceMapper {
protected Object getObjectFromStorage(OID oid) {
 ResultSet rs =
 RDBOperations.getInstance().getProductSpecificationData(oid);
 ProductSpecification ps = new ProductSpecification();
 ps.setPrice(rs.getDouble("PRICE"));
 // here's the essence of it
 String manufacturerForeignKey = rs.getString("MANU_OID");
 OID manuOID = new OID(manufacturerForeignKey);
 ps.setManufacturer(new ManufacturerProxy(manuOID));
}
```

# 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.**