DISEÑO ORIENTADO A OBJETOS

Prof. Esp. Ing. Agustín Fernandez

Patrón DAO

- **Conceptos Previos:**
  - Nace con la definición de los EJB dentro de la arquitectura J2EE cuando todavía esta tecnología era inmadura.
  - Se encuentra dentro de la definición de los J2EE Core Patterns (de Junio de 2003).
  - Apunta a resolver el problema de “mezclar” la lógica de negocio con la forma de acceder y manipular los diferentes medios de almacenamiento que pueden utilizarse en una aplicación empresarial.
  - Crea una capa de abstracción que “envuelve” el acceso y la manipulación de los datos de una aplicación.

Patrón DAO

- **Conceptos Previos (Cont.)**
- ¿Qué es un EJB?
  - EJB o Enterprise JavaBeans son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE (ahora JEE) de Oracle Corporation (inicialmente desarrollado por Sun Microsystems). Su especificación detalla cómo los servidores de aplicaciones proveen objetos desde el lado del servidor, que son, precisamente, los EJB.
  - Para trabajar con ellos nos basamos en un patrón Domain Model. Esto implica identificar todas las entidades de negocio y transformarlas en clases, de las cuales luego se generan instancias. Estas clases contienen cierta lógica del negocio o dominio. (Son clases especializadas)

Patrón DAO

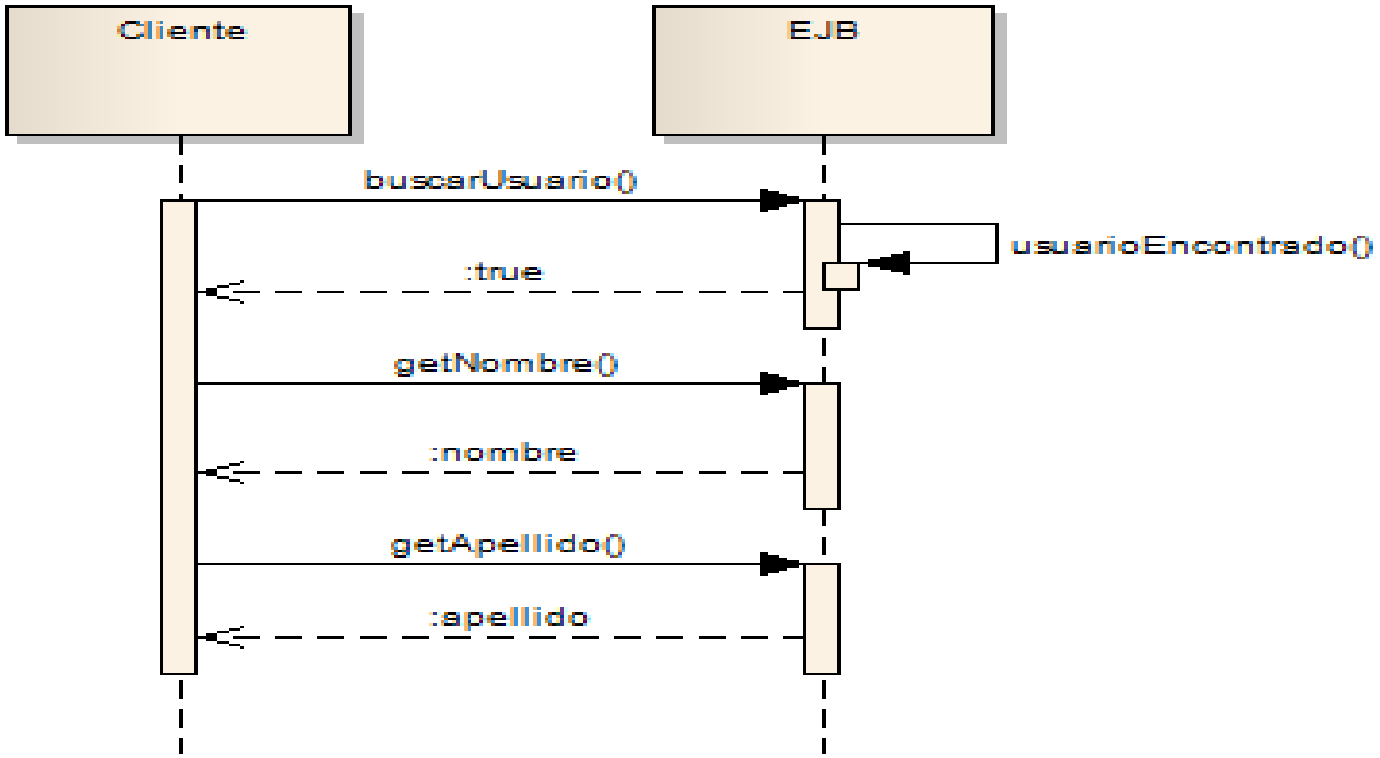
- **Conceptos Previos (Cont.)**
- ¿Qué es un EJB? (Cont.)
  - Los EJB's se encuentran en un contenedor encargado de generar las instancias de las entidades de negocio. Este contenedor entrega una instancia independiente a cada cliente que la solicite.
  - De esta manera, **cualquiera podría utilizar la lógica de negocio que esta almacenada en los EJB's.**

Patrón DAO

- **Conceptos Previos (Cont.)**
- ¿Qué relaciona EJB con DAO?
  - Como los EJB están basados en un patrón Domain Model, deben tener comportamiento y propiedades de lo que están modelando, así, supongamos que tenemos un EJB que modela la entidad Usuario, un cliente hace una petición hacia este para obtener los datos de un usuario especifico ¿qué pasaría? Simplemente se debería cargar el EJB con los datos del usuario que se esta buscando.
  - Si el objeto esta en el contenedor, para traerme sus atributos debería de hacer peticiones por cada atributo, algo similar a lo siguiente:

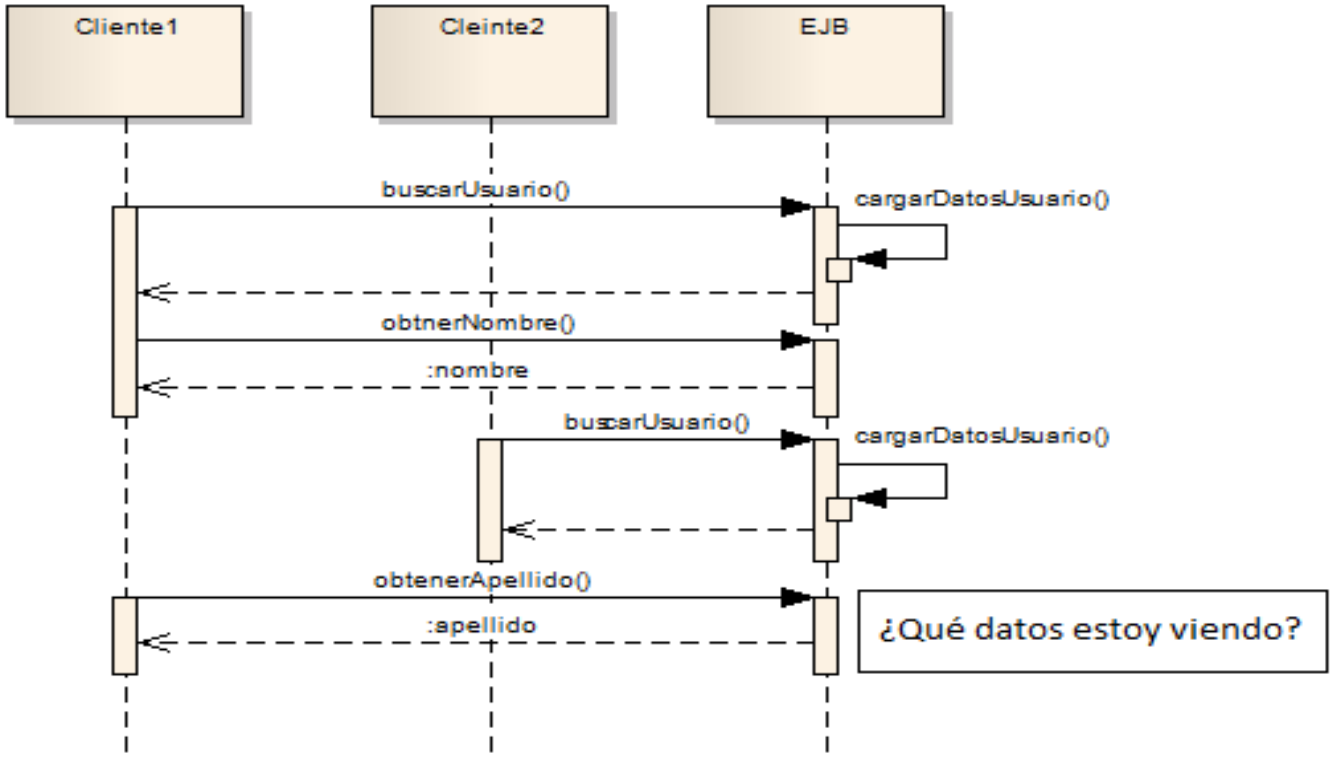
Patrón DAO

- **Conceptos Previos (Cont.)**
- ¿Qué relaciona EJB con DAO? (Cont.)
  - Muchas peticiones por la red hacia el EJB.



Patrón DAO

- **Conceptos Previos (Cont.)**
- ¿Qué relaciona EJB con DAO? (Cont.)
  - Manejar concurrencia para accesos simultáneos al EJB.



Patrón DAO

- **Conceptos Previos (Cont.)**
- ¿Qué relaciona EJB con DAO? (Cont.)
  - Para resolver las cuestiones antes mencionadas surgió el concepto de **Transfer Object**.
  - **Transfer Object**:
    - Guardar toda la información con la que debería de cargarse el EJB en un objeto independiente.
    - Es un objeto que solo sirve como un mecanismo para transportar información de un punto a otro y que no tiene comportamiento.
    - Es decir un objeto que solo contiene los atributos de la entidad Usuario y devuelve este objeto al cliente, así la red se estresará menos y no habrá problema de concurrencia.

Patrón DAO

- **Conceptos Previos (Cont.)**
- ¿Qué relaciona EJB con DAO? (Cont.)
  - Ya sabemos lo que es un **Transfer Object**, esto nos permitirá entender mejor la estructura del **patrón DAO**.

Patrón DAO

- Contexto:
  - El acceso a los datos varía dependiendo de la fuente de los datos.
  - El acceso al almacenamiento persistente, como una base de datos, varía en gran medida dependiendo del tipo de almacenamiento (bases de datos relacionales, bases de datos orientadas a objetos, ficheros planos, etc.) y de la implementación del vendedor.

Patrón DAO

- Problema:
  - Muchas aplicaciones de la plataforma Java y J2EE en el mundo real necesitan utilizar datos persistentes en algún momento.
  - Este almacenamiento persistente se implementa utilizando diferentes mecanismos y hay marcadas **diferencias en los APIS utilizados para acceder a** esos **mecanismos de almacenamiento diferentes**.
  - Incluso dentro de un entorno RDBMS, la sintaxis y formatos de las sentencias SQL podrían variar dependiendo de la propia base de datos en particular.

Patrón DAO

- Problema: (Cont.)
  - Entonces **fuentes de datos dispares** ofrecen retos a la aplicación y potencialmente **pueden crear una dependencia directa entre el código de la aplicación y el código de acceso a los datos**.
  - Dichas dependencias de código en los componentes hace **difícil y tedioso migrar la aplicación de un tipo de fuente de datos a otro**.

Patrón DAO

- Causas:
  - Los APIs para almacenamiento persistente varían dependiendo del vendedor del producto.
  - Otras fuentes de datos podrían tener APIS que no son estándar y/o propietarios.
  - Estos APIs y sus capacidades también varían dependiendo del tipo de almacenamiento: bases de datos relacionales, bases de datos orientadas a objetos, documentos XML, ficheros planos, etc.
  - Hay una falta de APIs uniformes para corregir los requerimientos de acceso a sistemas tan dispares.
  - Los componentes necesitan ser transparentes al almacenamiento persistente real o la implementación de la fuente de datos para proporcionar una migración sencilla a diferentes tipos de fuentes de datos.

Patrón DAO

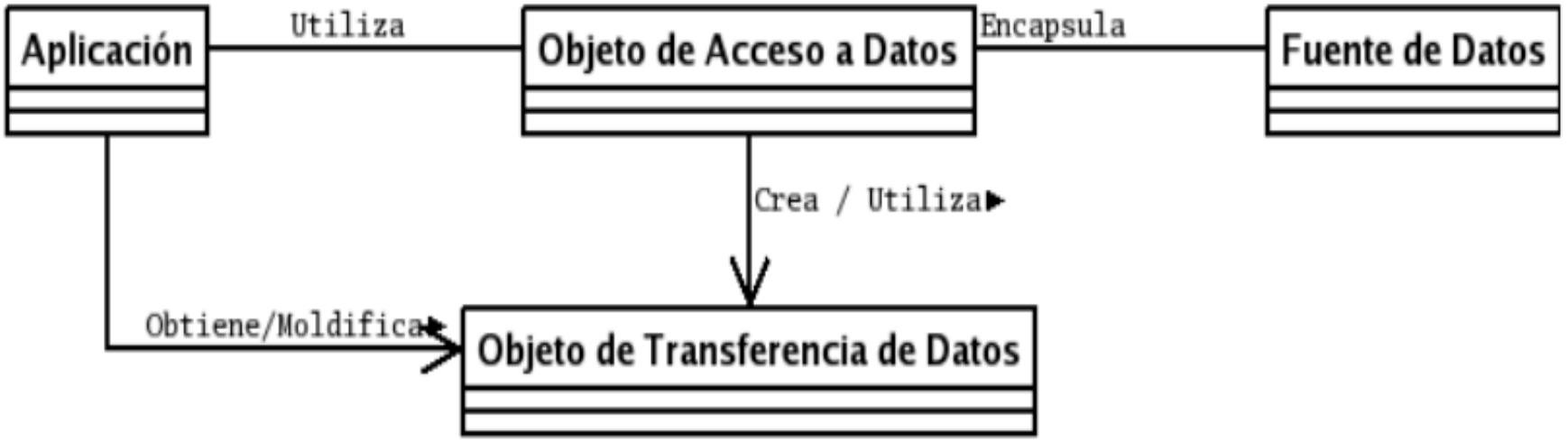
- Solución:
  - Utilizar un Data Access Object (DAO) para abstraer y encapsular todos los accesos a la fuente de datos.
  - El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos.
  - El DAO implementa el mecanismo de acceso requerido para trabajar con la fuente de datos.
  - Esta fuente de datos puede ser un almacenamiento persistente como una RDMBS, un servicio externo como un intercambio B2B, un repositorio LDAP, sockets de bajo nivel, etc.
  - Los componentes de negocio que tratan con el DAO utilizan un interface simple expuesto por el DAO para sus clientes.

Patrón DAO

- Solución: (Cont.)
  - El DAO oculta completamente los detalles de implementación de la fuente de datos a sus clientes.
  - Como el interface expuesto por el DAO no cambia cuando cambia la implementación de la fuente de datos subyacente, **este patrón permite al DAO adaptarse a diferentes esquemas de almacenamiento sin que esto afecte a sus clientes o componentes de negocio.**
  - **Esencialmente, el DAO actúa como un adaptador entre el componente y la fuente de datos.**
  -

Patrón DAO

- Estructura:



Patrón DAO

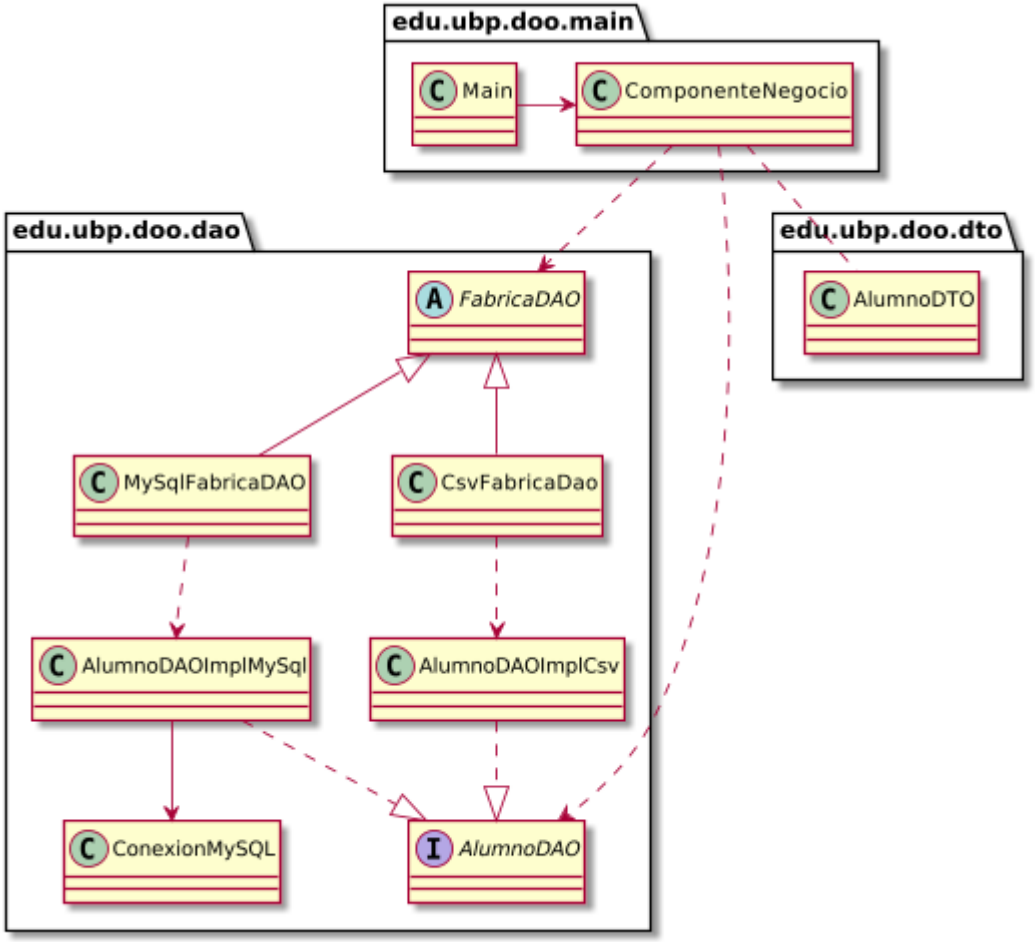
- Participantes:
  - BusinessObject (Aplicación): es el objeto que requiere el acceso a la fuente de datos para obtener y almacenar datos.
  - DataAccessObject: es el objeto principal de este patrón. DataAccessObject abstrae la implementación del acceso a datos subyacente al BusinessObject para permitirle un acceso transparente a la fuente de datos. El BusinessObject también delega las operaciones de carga y almacenamiento en el DataAccessObject.

Patrón DAO

- Participantes: (Cont.)
  - DataSource: representa la implementación de la fuente de datos. Una fuente de datos podría ser una base de datos como un RDBMS, un OODBMS, un repositorio XML, un fichero plano, etc. También lo pueden ser otros sitemas (mainframes/Gubernamentales/Heredados), servicios (servicio B2B), o algún tipo de repositorio (LDAP).
  - TransferObject: representa un Transfer Object utilizado para el transporte de datos. DataAccessObject podría utilizar un Transfer Object para devolver los datos al cliente. El DataAccessObject también podría recibir datos desde el cliente en un Transfer Object para actualizar los datos en la fuente de datos.

Patrón DAO

- **Ejemplo de Implementación**



Patrón DAO

- Para tener en cuenta:
  - DAO es un patrón venerable que logró perdurar más allá de J2EE.
  - El principal problema en J2EE fue el insuficiente poder de CMP 2.0 (Container Managed Persistence), las capacidades de consulta limitadas (no había acceso a SQL nativo ni consultas dinámicas)
  - DAO fue una muy buena idea para encapsular el acceso a la base de datos.
  - Lo que DAO no tardó en llegar a ser, en la práctica, era un reemplazo para los EJB.
  - La aplicación más común, como vimos, asigna una sola tabla a una clase DAO que implementa métodos de búsqueda que devuelven colecciones de un objeto de dominio.

Patrón DAO

- Para tener en cuenta: (Cont.)
  - El problema aquí es que tendremos suerte si el autor del DAO pensó en la consulta que se desea realizar.
  - En este caso estamos dentro de una estrategia de implementación sin salida para consultar datos.
  - También muchas veces la combinación Factory - DAO, va de la mano con el patrón de pensamiento: **“si ya no es tan útil, seguí tratando hasta tener razón”**.

Patrón DAO

- Para tener en cuenta: (Cont.)
  - En la actualidad en EJB 3 (Java EE 5 en adelante) ya no hay necesidad de utilizar el API de bajo nivel de JDBC para acceder a la base de datos.
  - Se puede usar genéricos, SQL nativo, buscar no sólo objetos persistentes, sino también objetos de transferencia de datos y tipos de datos primitivos.
  - JPA ya viene con EntityManager que proporciona funcionalidades genéricas de acceso a datos. Su uso es muy simple.

Patrón DAO

- Para tener en cuenta: (Cont.)
  - Si comparamos veremos que para aplicar DAO tenemos que implementar lo siguiente:
    - DAO-Interface
    - DAO-Implementation
    - DAO-Factory
  - En cambio el EntityManager se inyecta directamente en el Bean:
    - @Stateless
    - public class CustomerMgrBean implements CustomerMgr{
    - @PersistenceContext
    - private EntityManager em;

Patrón DAO

- Para tener en cuenta: (Cont.)
  - Por esto y mucho más cuando vayamos a aplicar DAO debemos tener en cuenta lo que dice Adam Bien en su libro "Real World Java EE Patterns--Rethinking Best Practices":
    - The vendor of your database changes?
    - You will have to replace your relational database with LDAP, flat files or object oriented storage?
    - You will have to replace JPA with iBatis or even plain JDBC?
    - You will have to replace your local storage with remote service calls?
  - Si la respuesta es sí, entonces DAO es la opción.
  - Sino deberías pensar en usar alguna otra solución.

Patrón DAO

- Para tener en cuenta: (Cont.)
  - “La arquitectura (incluso en la vida real) es todo acerca de probabilidades. Si algo es probable que suceda, reaccionamos a ello. Si algo es más bien poco probable, tenderemos a ignorarlo”. – *Adam Bien.*