Patrón de Diseño: Data Access Object (DAO)

Temas selectos de programación
Profesor: Jose C Aguilar Canepa

Alumno: Herrera Pacheco Gerardo Isidro
ISC 68612
8vo semestre
12/03/2025

Problemática

Problemas comunes al acceder a datos:

- Las aplicaciones necesitan usar datos persistentes que pueden residir en diferentes fuentes (bases de datos relacionales, ficheros XML, JSON, repositorios LDAP, servicios externos, etc.).

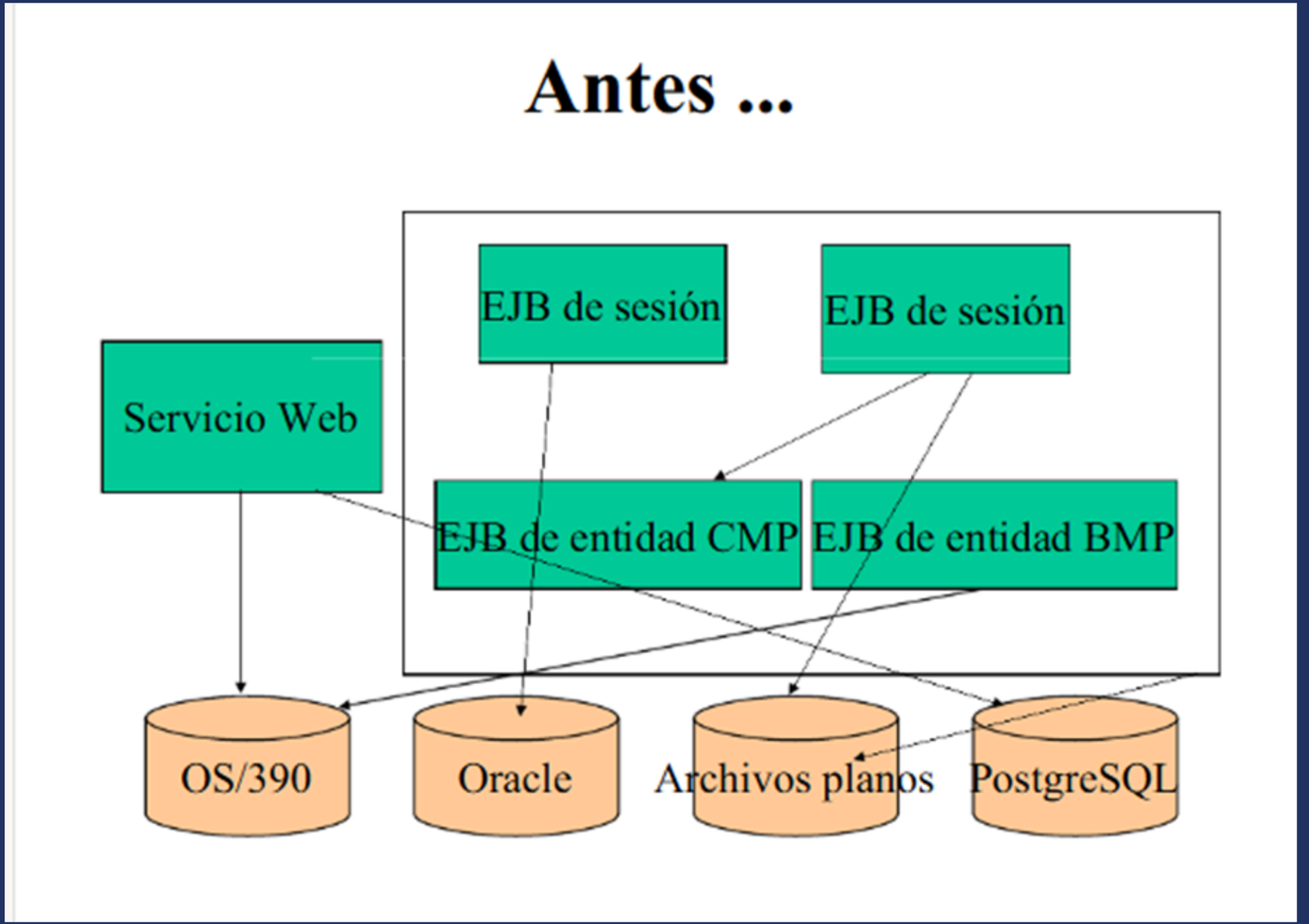
- Las diferencias entre los mecanismos de almacenamiento y las APIs de acceso generan acoplamiento entre la lógica de negocio y la lógica de acceso a datos.

Problemática

Problemas comunes al acceder a datos:

- El formato e implementación de la fuente de datos pueden variar.
- Si el Código de acceso a datos esta mezclado con lógica de negocio.
- Puede haber dificultad para cambiar o agregar nuevas fuentes de datos.

Problemática



Solución propuesta por DAO

Separación clara entre lógica de negocio y acceso a datos.

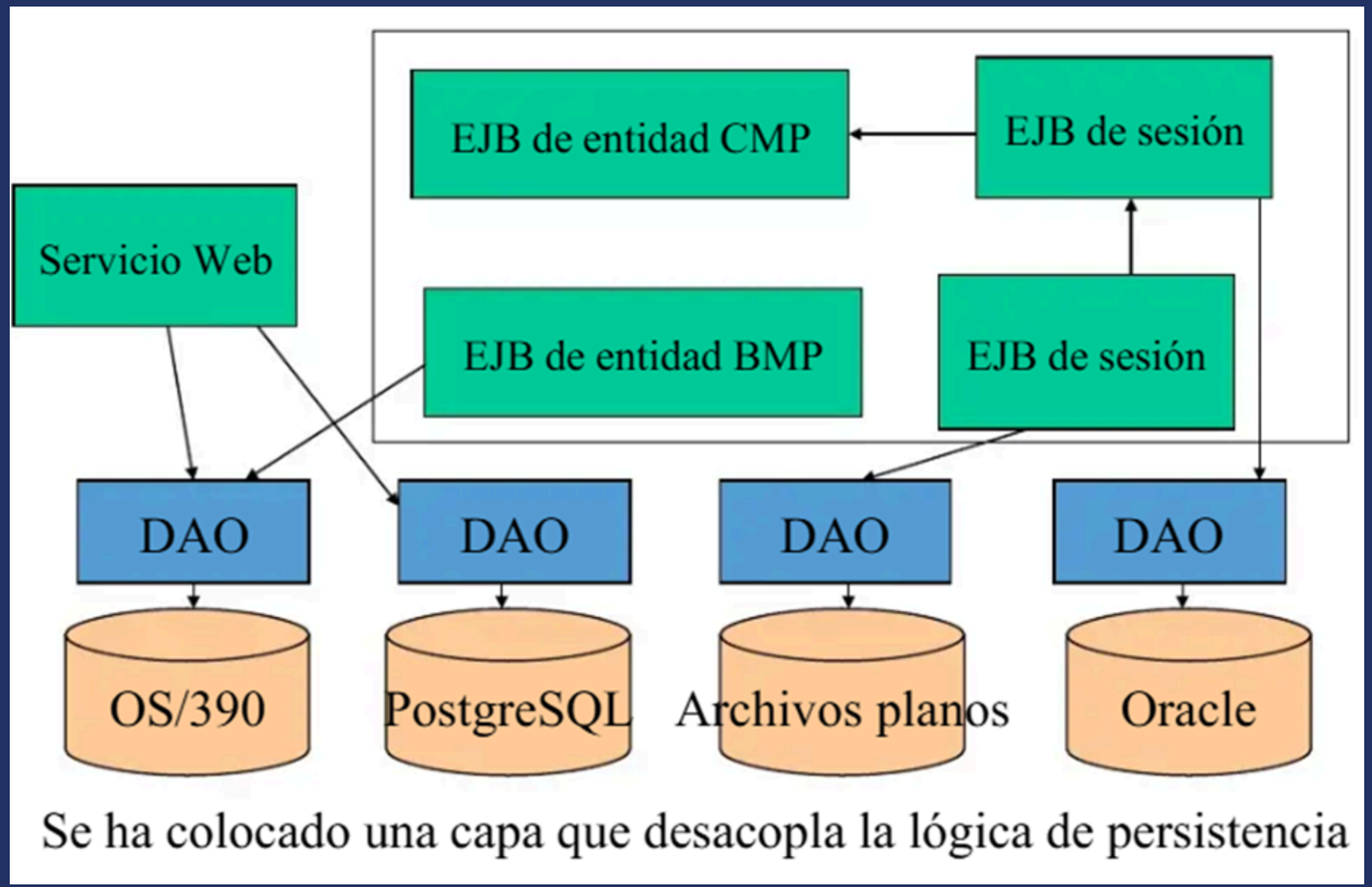
DAO proporciona normalmente métodos para CRUD (Create, Read, Update, Delete).

Encapsulación de detalles técnicos de la fuente de datos.

El DAO gestiona la conexión con la fuente de datos para obtener y almacenar datos.

Solución propuesta por DAO

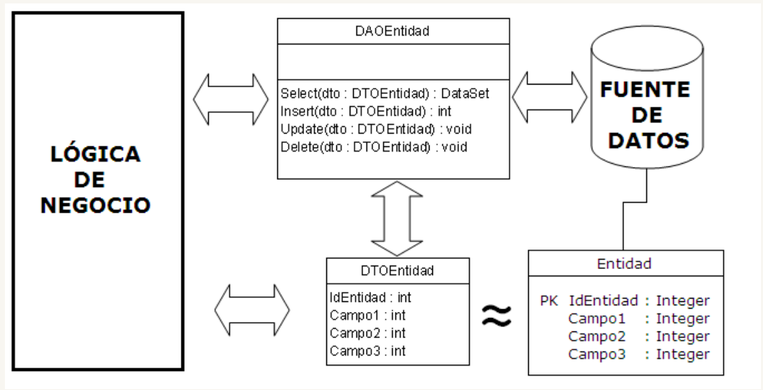
Después..



¿Qué es el patrón DAO?

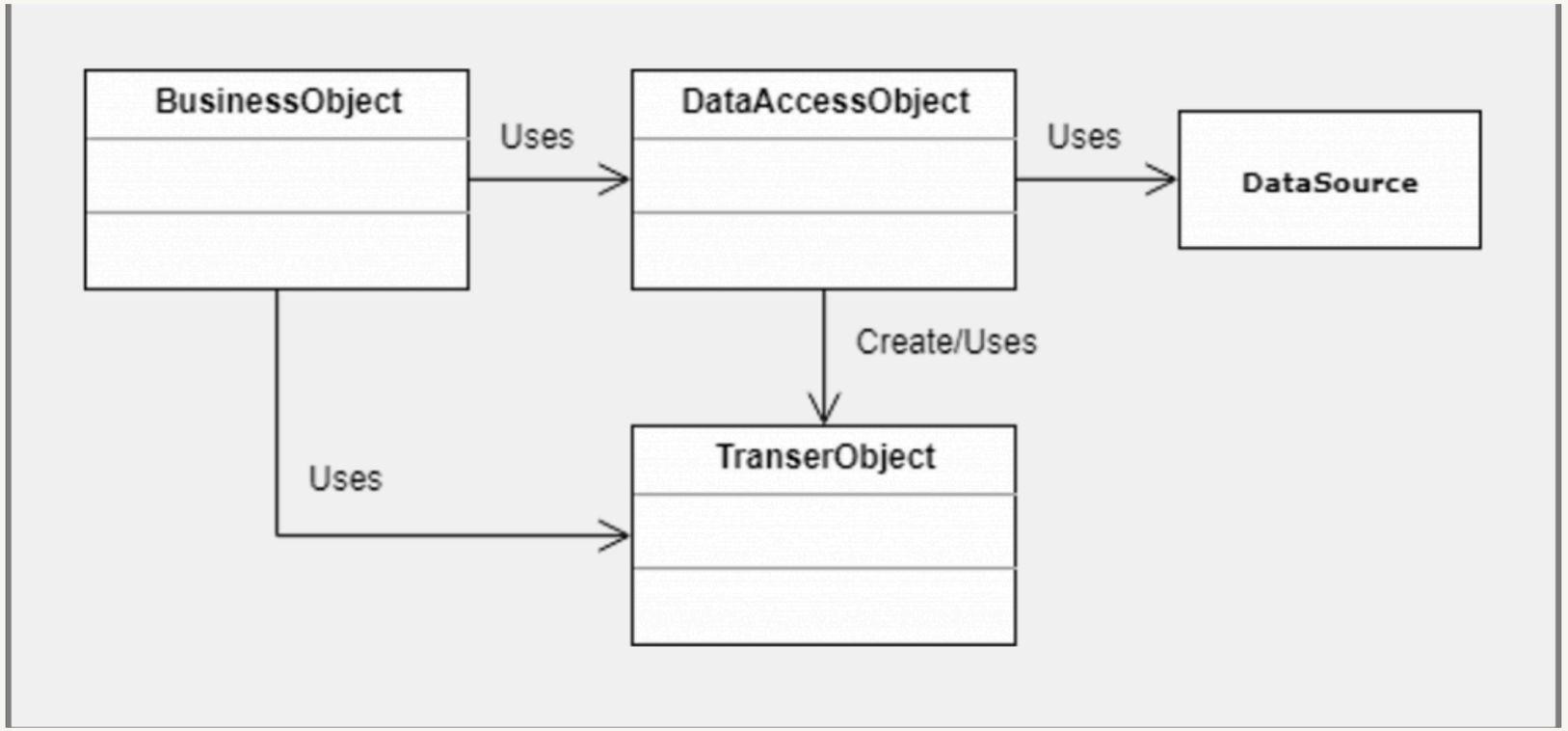
Método de estructuración para separar la lógica de negocio y el acceso a datos. Es un modelo para abstraer complejidades de almacenamiento y recuperación de datos.

Si cambia la fuente de datos, solo es necesario modificar la implementación del DAO, sin afectar al resto de la aplicación.



¿Abstracción, encapsulamiento?

Componentes del patrón DAO



¿Cómo se relacionan?

BusinessObject

- Representa un objeto que contiene la lógica de negocio.
- No interactúa directamente con la fuente de datos.

DataAccessObject

- Objeto que encapsula el acceso a datos.
- Proporciona métodos CRUD.

Ejemplos

Java

```java
public class UserService {
    private UserDao userDao = new UserDao();

    public void addUser(User user) {
        userDao.save(user);
    }
}
```

Java

```java
public class UserDao implements Dao<User> {
    @Override
    public void save(User user) {
        // Lógica para guardar en la base de datos
    }
}
```

TransferObject

- Objeto de transferencia que actúa como portador de datos
- Se utiliza para transmitir datos entre un cliente y un servidor, o entre varios niveles de una aplicación.

DataSource

- Fuente de datos.
- Puede ser una base de datos, servicio web, archivo XML, etc.

Ejemplos

Java

```java
public class User {
    private String name;
    private String email;

    // Getters y setters
}
```

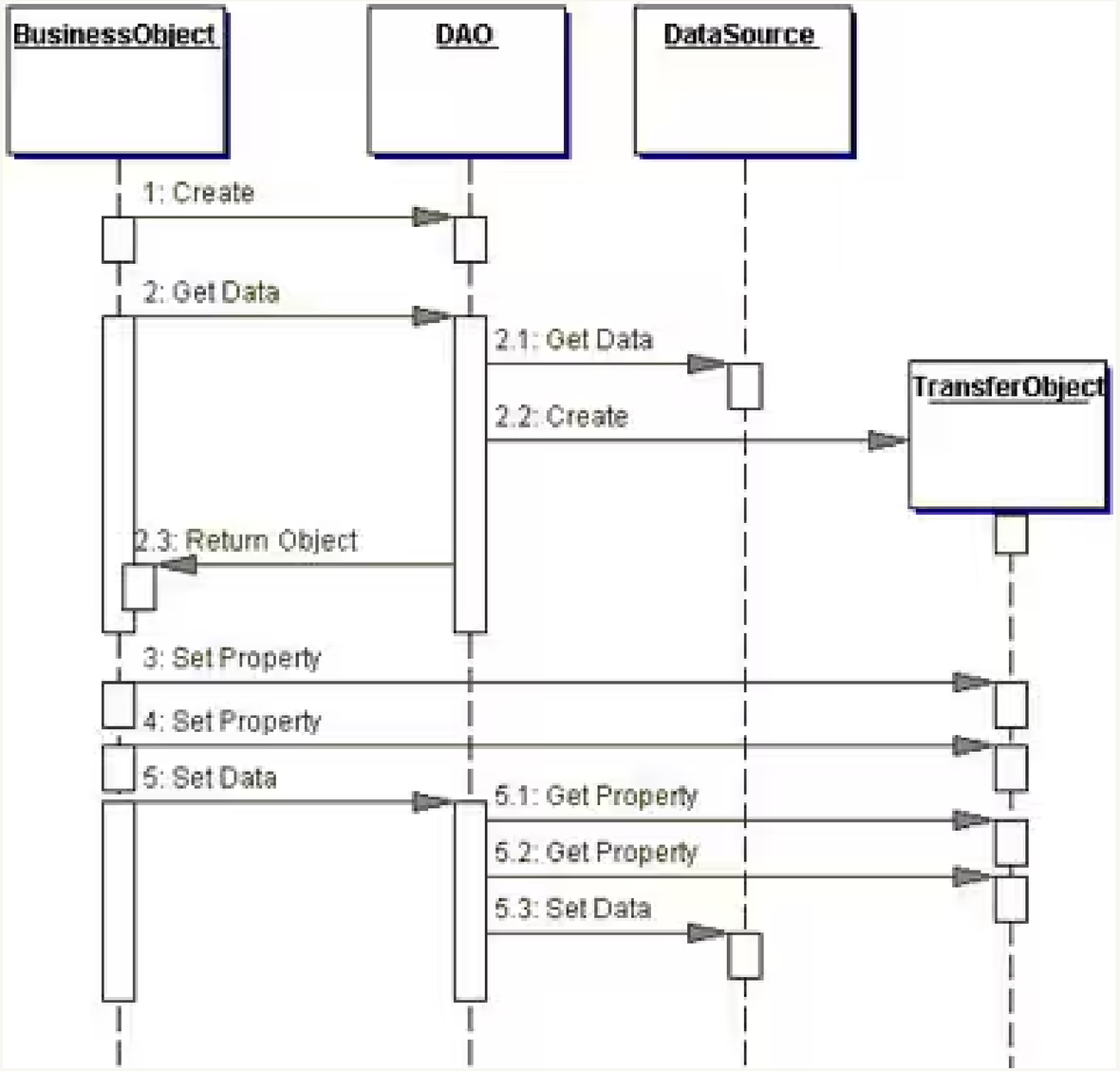
Java

```java
DataSource dataSource = new MysqlDataSource();
```

Cómo interactúan los componentes entre sí:

- El objeto de negocio se comunica con el objeto de movimiento para facilitar el movimiento de datos.
- El DAO interactúa con el objeto de negocio y el objeto de transferencia. El sistema utiliza el objeto de transferencia para facilitar el intercambio de datos entre la base de datos y la aplicación. También puede colaborar con los objetos de negocio para transformar los datos del formato de transferencia a la representación interna que utiliza la aplicación.
- El DAO utiliza la fuente de datos para adquirir las conexiones de base de datos necesarias para realizar consultas y modificaciones.

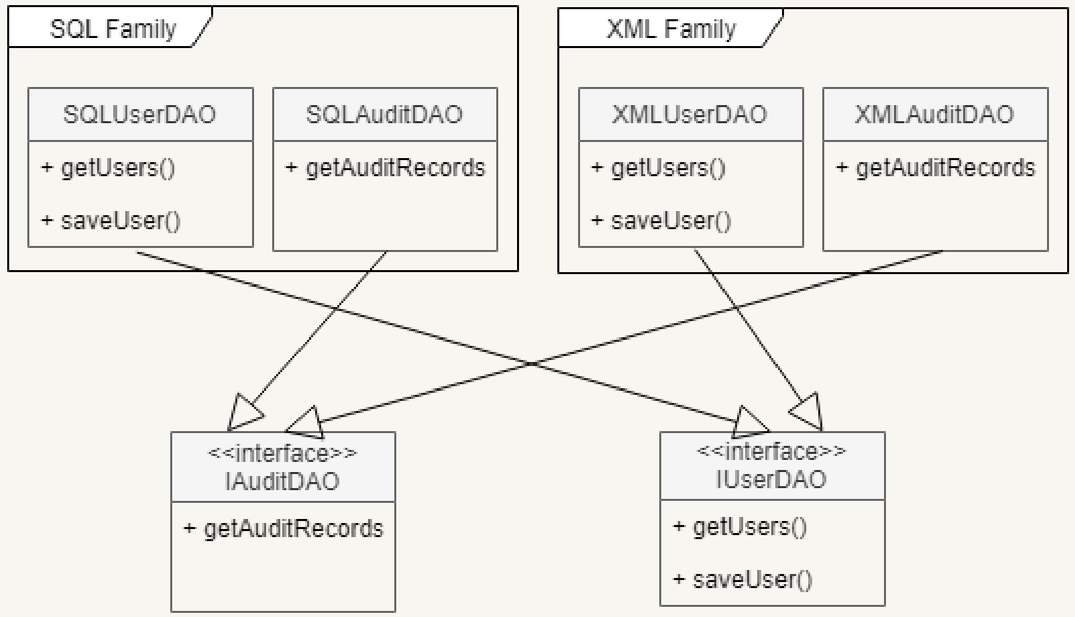
Diagrama de secuencia



Abstract Factory

¿Qué pasa si tenemos múltiples fuentes de datos (SQL, NoSQL, XML)?

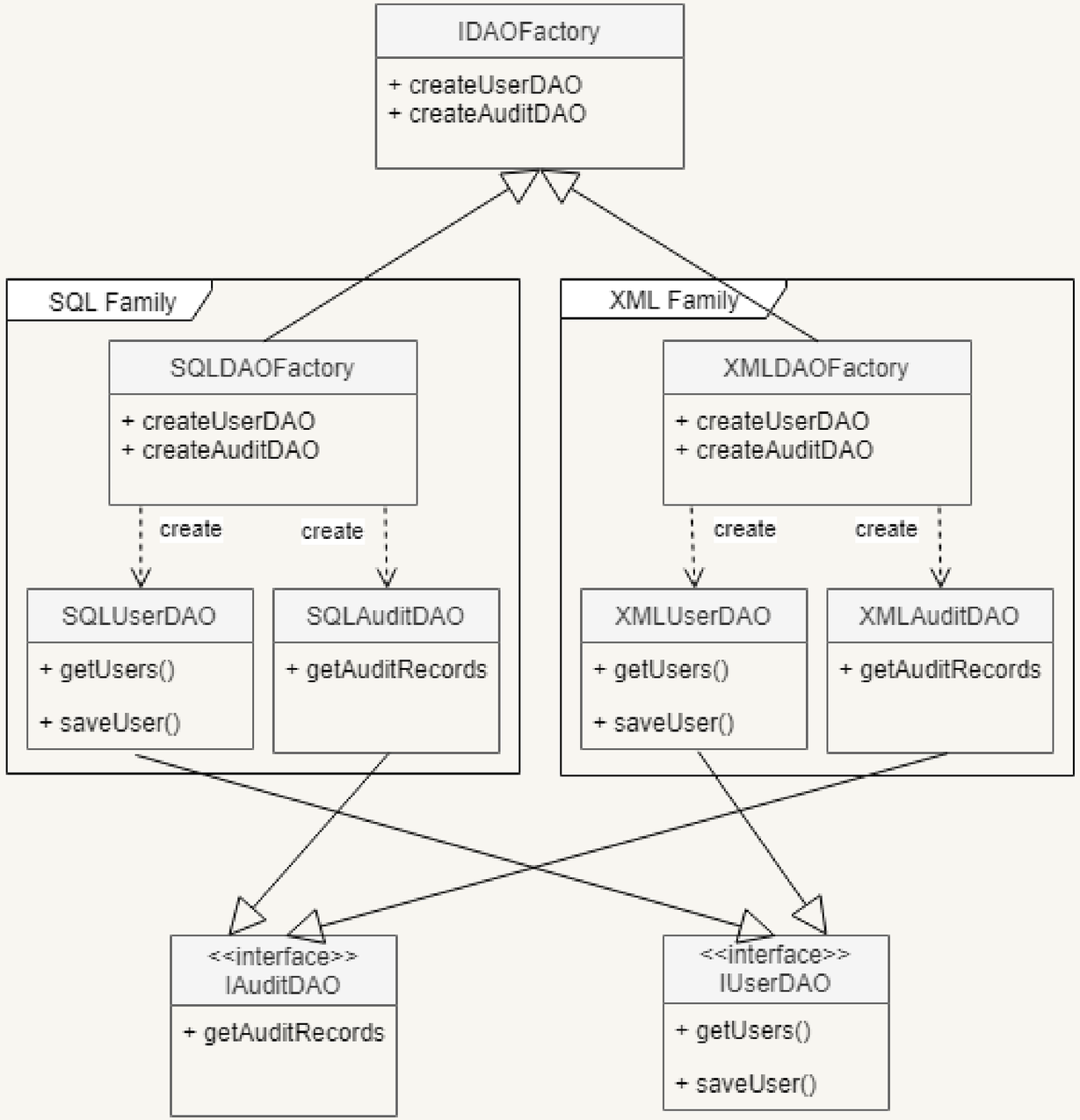
- DAO tradicional no puede adaptarse fácilmente
- Abstract Factory permite crear DAOs específicos según la fuente de datos.



¿Qué logra esto?

Abstract Factory

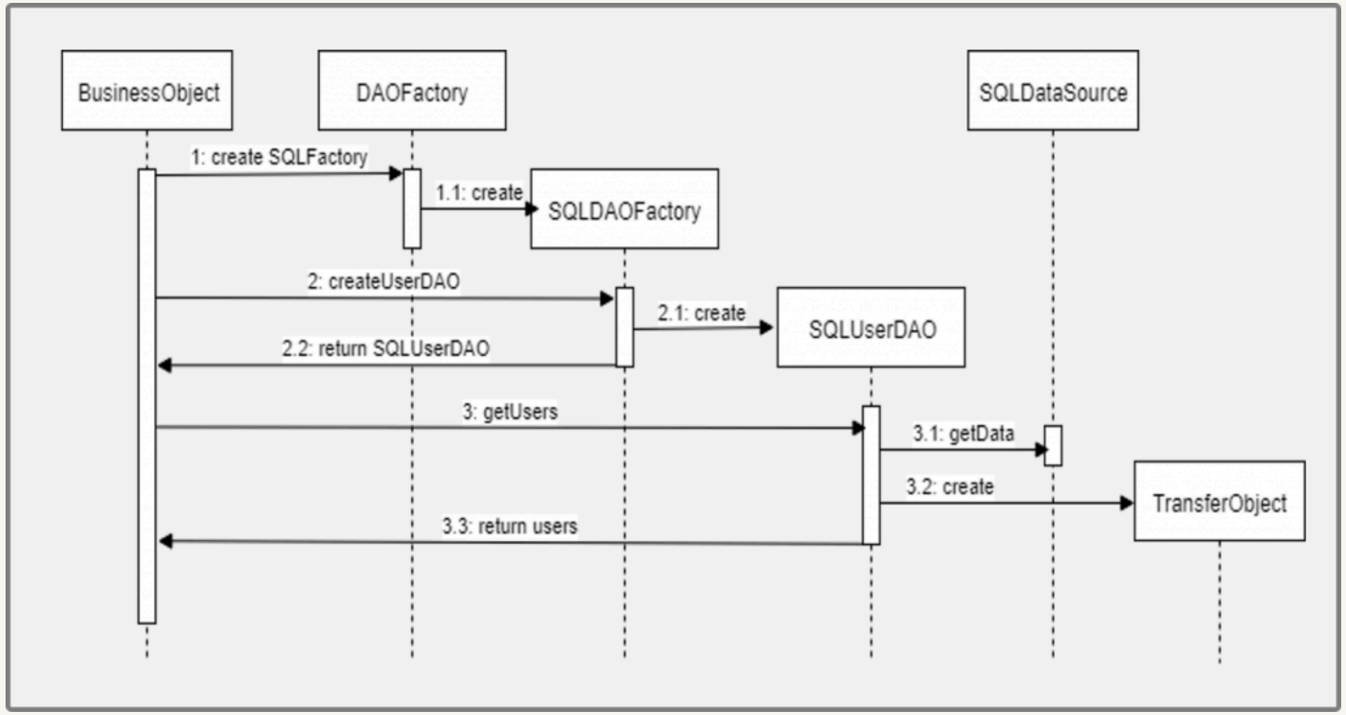
En esta nueva configuración, podemos ver que tenemos un Factory para cada familia, y los dos factorys implementan una interfaz en común, adicional, tenemos la interface IDAOFactory necesaria para que el factory de cada familia implementen una interface en común.



Secuencia de ejecución

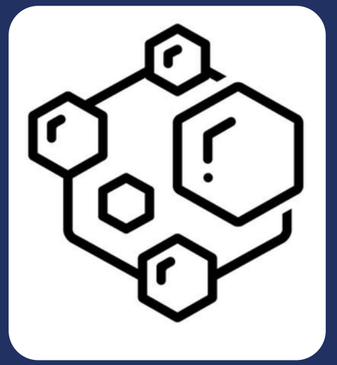
1. El BusinessObject solicita la creación de un DAOFactory para SQL
    - El DAOFactory crea una instancia de la clase SQLDAOFactory y la retorna
2. El BusinessObject solicita al SQLDAOFactory la creación del SQLUserDAO para interactuar con los usuarios.
    - El SQLDAOFactory crea una nueva instancia del SQLUserDAO
    - El SQLDAOFactory retorna la instancia creada del SQLUserDAO
3. El BusinessObject solicita el listado de todos los usuarios registrados al SQLUserDAO
    - El SQLUserDAO recupera los usuarios del SQLDataSource
    - El SQLUserDAO crea un TransferObject con los datos recuperados del paso anterior.
    - El SQLUserDAO retorna el TransferObject creado en el paso anterior.

Diagrama de secuencia



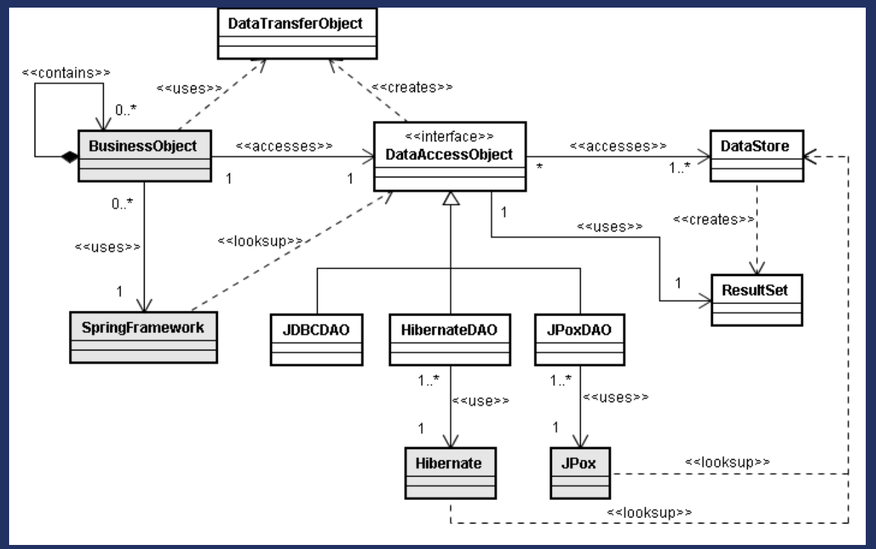
Beneficios del patrón DAO

B	La lógica de negocio y la lógica de acceso a datos están separadas
D	Código limpio y reutilizable.
O	Facilidad de mantenimiento.
U	Escalabilidad y flexibilidad.
U	Mayor facilidad para realizar pruebas unitarias.



Desventajas del patrón DAO

B	Añade una capa adicional.
O	Dependencia en una jerarquía de clases.
U	Posible sobrecarga si solo hay una fuente de datos simple.
U	Complejidad en la implementación inicial.



Referencias

Blancarte Iturralde, O. J. (2016). Introducción a los patrones de diseño (1ra ed., versión 1.5). Autor.

Core J2EE Patterns - Data Access Object. (s. f.). https://www.oracle.com/java/technologies/dataaccessobject.html

Data access object pattern. (s. f.). https://www.tutorialspoint.com/design_pattern/data_access_object_pattern.htm

Mastering DAO and DTO Patterns in Java Development. (2024, 18 junio). https://www.index.dev/blog/what-are-dao-and-dto-in-java-explained