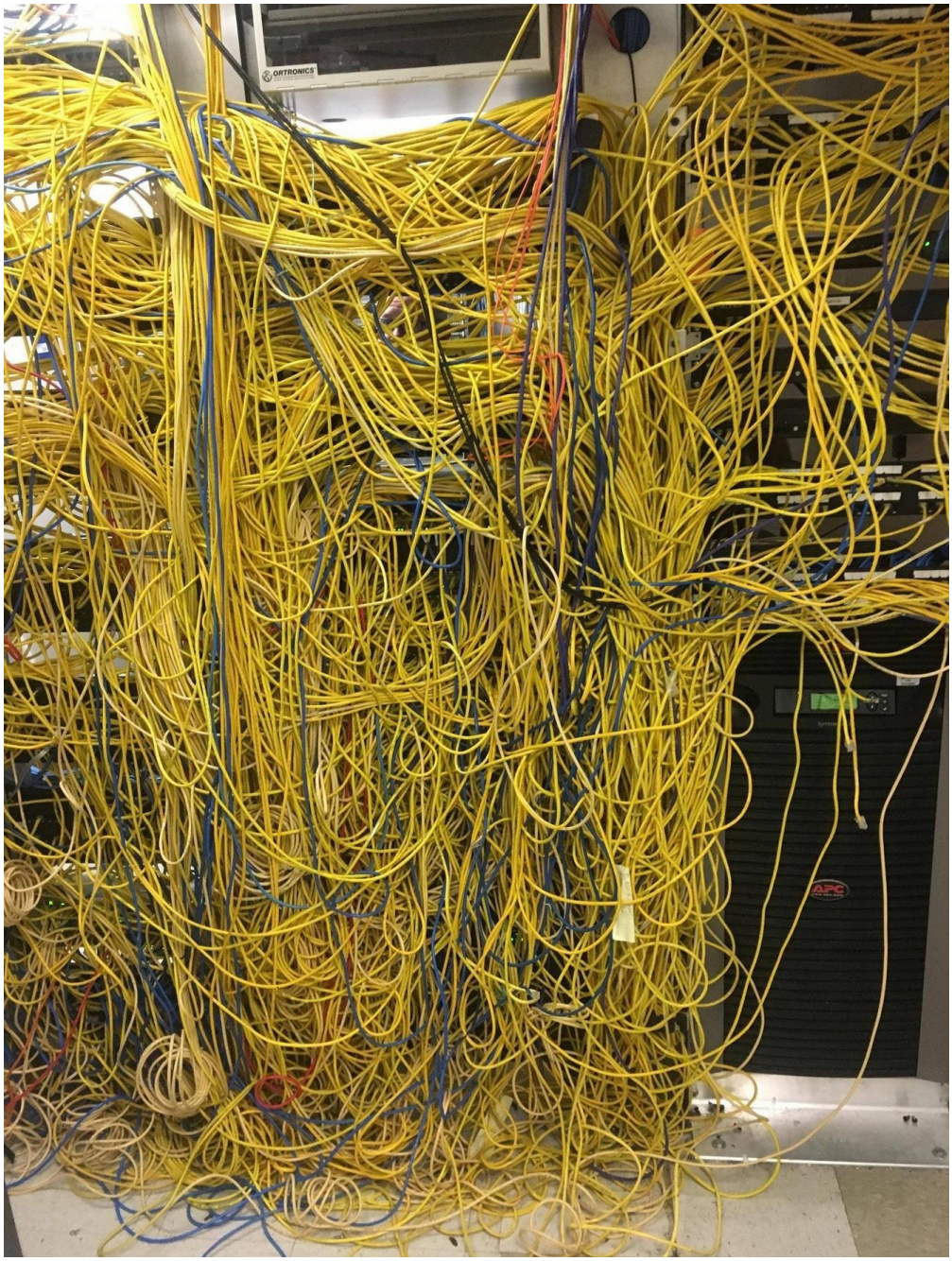
El patrón DAO. MySQL y JDBC (II)

El patrón DAO

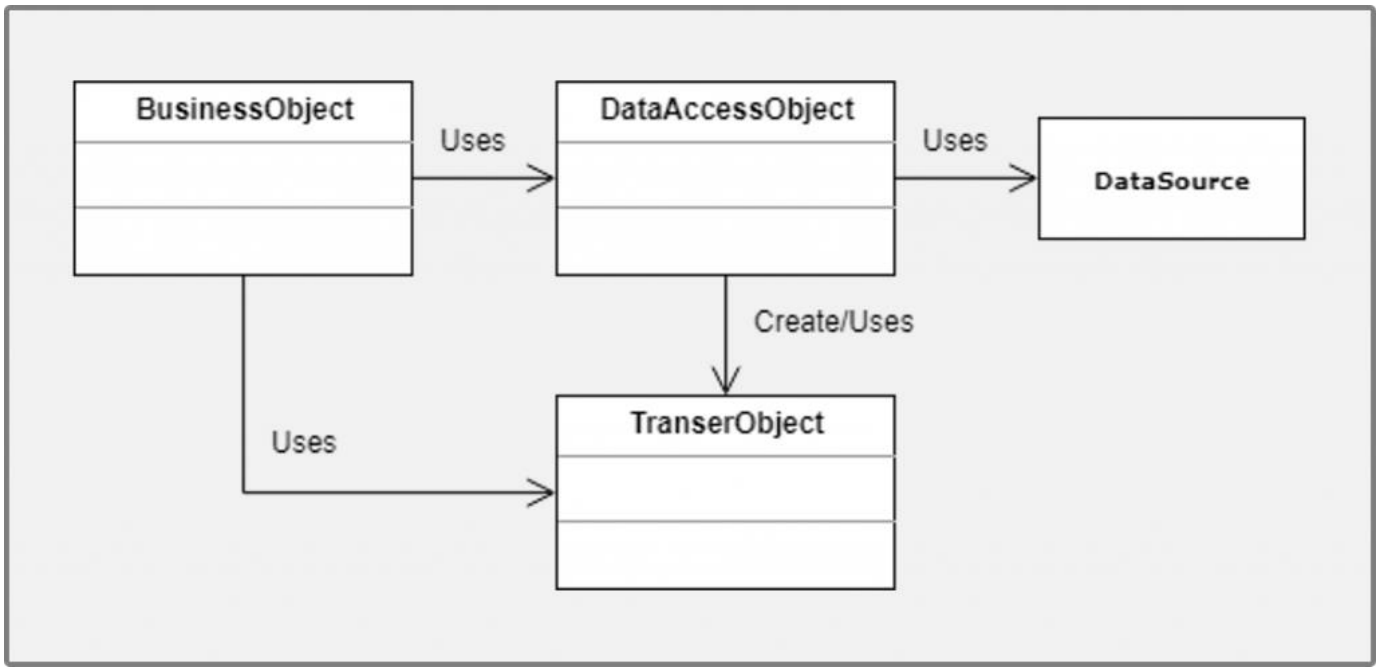
Una vez que tenemos los rudimentos para gestionar una base de datos, llega el momento de organizar el código. Existen diferentes aproximaciones para evitar lo siguiente:

El impulso inicial

El impulso inicial de todo programador novel es insertar consultas donde le hace falta. Conforme el código va aumentando, y van apareciendo bugs o se necesita hacer cambios, el código tiene este aspecto:



Para evitar esto, existen varios patrones (formas de estructurar el código). El primero y más sencillo (no por ello menos efectivo) es el patrón DAO (Data Access Object)

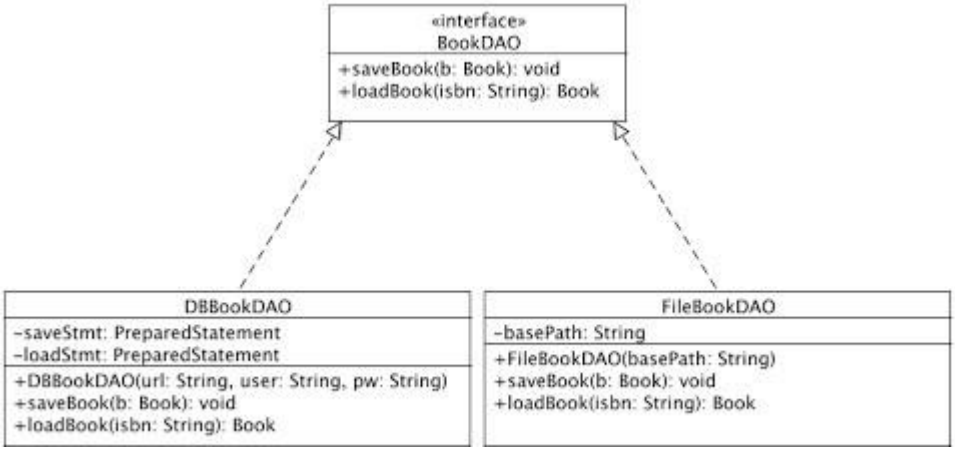


- **BusinessObject** es una clase de la capa de negocio, que son las funciones principales de la aplicación: procesamiento de datos, coordinación de usuarios y administración de recursos externos. Por ejemplo, en el caso de un programa de nóminas, podría serla clase **GestionNominas**.
- **TransferObject** es una clase también de la capa de negocio, normalmente un POJO (Plain Old Java Object). Por ejemplo, en el caso de un programa de nóminas, podría ser la clase **Empleado**.
- **DataAccessObject** (DAO) es una clase encargada de interactuar con el almacén de datos (en nuestro caso una base de datos, aunque puede ser también un archivo u otra fuente de datos), para escribir, leer, modificar y borrar.

El objetivo de este patrón es que el objeto *BusinessObject* no necesite saber qué tipo de almacén de datos está utilizando.

La arquitectura en capas clásica dejaría *BusinessObject* y *TransferObject* en la capa de *lógica de negocio* y *DataAccessObject* en la capa de persistencia.

Independizar la lógica de negocio del almacén de datos



Ahora observa las siguientes líneas:

```
BookDAO bookDAO1 = new DBBookDAO;
BookDAO bookDAO2 = new FileBookDAO;
```

Ambas variables son de clase BookDAO, pero su tipo es diferente. En ambos casos podríamos hacer:

```
bookDAO1.saveBook(libro);
bookDAO2.saveBook(libro);
```

Pero el resultado en cada caso es diferente. En un caso estamos guardando el libro en un fichero, y en el otro en una tabla de una base de datos.

> En la práctica se suele combinar el patrón DAO con el patrón *Factory* (que no hemos visto todavía), para que la independencia de la lógica de negocio y la capa de persistencia sea total. El aspecto de la instanciación de un objeto DAO en la lógica de negocio es el siguiente:
>
> ```
> ...
> BookDAO bookDAO = BookDAOFactory.crear(tipoAlmacen);
> ...
> ```
>
> Donde BookDAOFactory podría ser como sigue:
>
> ```
> public class BookDAOFactory {
>  private final static int XML = 1;
>  private final static int DB = 2;
>
>  public static BookDAO crear(int tipoAlmacen){
>         switch (tipoAlmacen){
>                 case XML:
>                         return new FileBookDAO();
>                 case DB:
>                         return new DBBookDAO();
>         }
>  }
> }
> ```