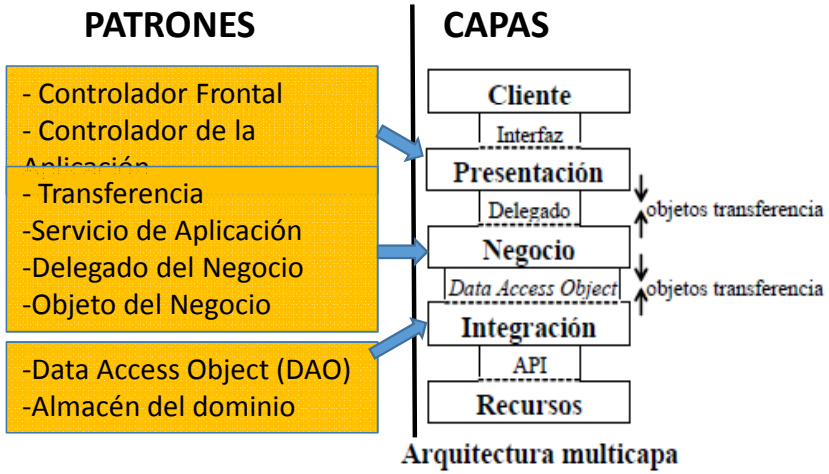
Arquitectura multicapa
Patrones relacionados

- Patrones relacionados:
  - Controlador frontal (capa de presentación)
  - Controlador de aplicación (capa de presentación)
  - Transferencia (capa de negocio)
  - *Data Access Object* (DAO) (capa de integración)
  - Servicio de aplicación (capa de negocio)
  - Delegado del negocio (capa de negocio)
  - Objeto del negocio (capa de negocio)
  - Almacén del dominio (capa de integración)

Arquitecturas Multicapa

Arquitectura multicapa
Patrones relacionados



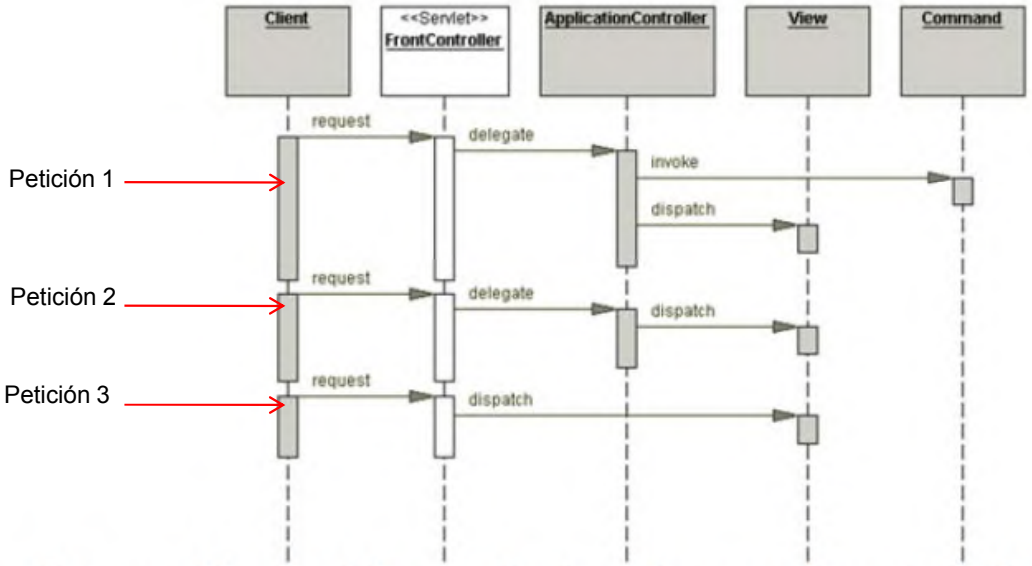
1- Patrón controlador frontal

- Propósito
  - Proporciona un punto de acceso para el manejo de las peticiones de la capa de presentación
- También conocido como
  - *Front controller*

1-Patrón controlador frontal

- Motivación
  - Se desea evitar lógica de control duplicada
  - Se desea aplicar una lógica común a distintas peticiones
  - Se desea separar la lógica de procesamiento del sistema de la vista
  - Se desea tener puntos de acceso centralizado y controlado al sistema

1-Patrón controlador frontal



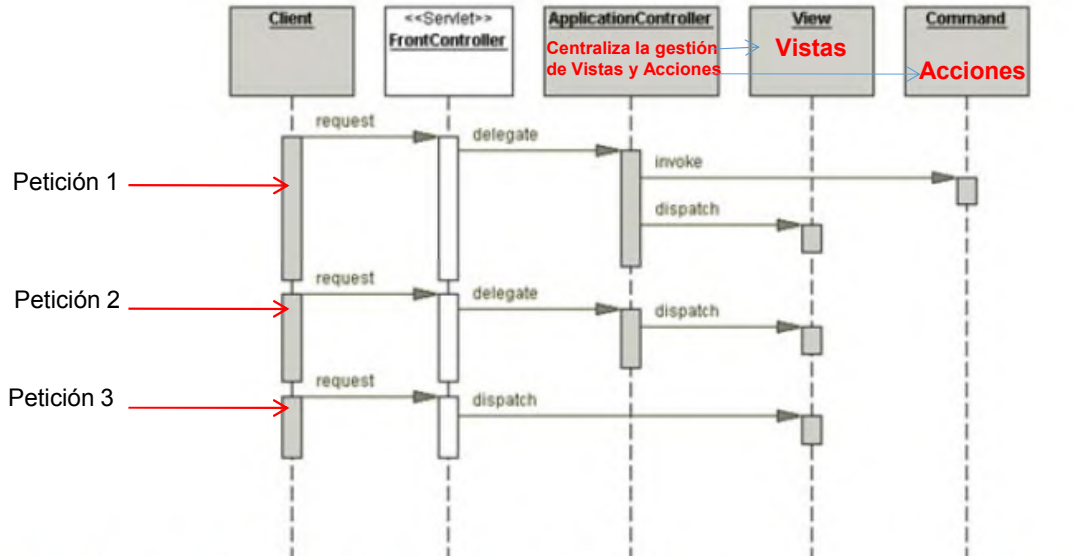
Interacción de objetos relacionados por el controlador frontal

- Consecuencias
  - Ventajas:
    - Centraliza el control
    - Mejora la gestión de la aplicación
    - Mejora la reutilización
    - Mejora la separación de roles
  - Inconvenientes
    - En aplicaciones grandes puede llegar a crecer mucho

2- Patrón controlador de aplicación

- Propósito
  - Se desea centralizar y modularizar la gestión de acciones y de vistas
- También conocido como
  - *Application controller*

2-Patrón controlador de aplicación



2-Patrón controlador de aplicación

- Motivación
  - Se desea reutilizar el código de gestión de vistas y acciones
  - Se desea mejorar la extensibilidad del manejo de peticiones (p.e. añadir casos de uso una aplicación incrementalmente)

2-Patrón controlador de aplicación

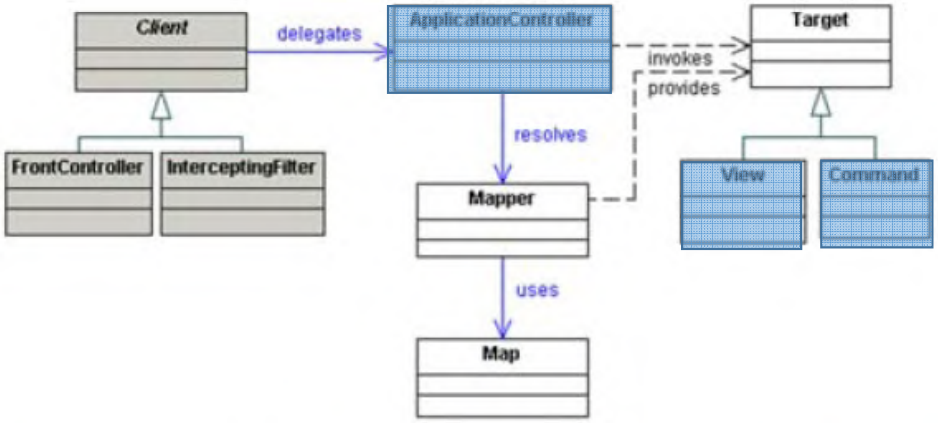
- Se desea mejorar la modularidad del código y la mantenibilidad, facilitando al extensión de la aplicación y la prueba del código de manejo de peticiones de manera independiente del contenedor web

2- Patrón controlador de aplicación

- Debe aplicarse cuando
  - Se quiera centralizar la recuperación e invocación de componentes de procesamiento de las peticiones, tales como comandos y vistas

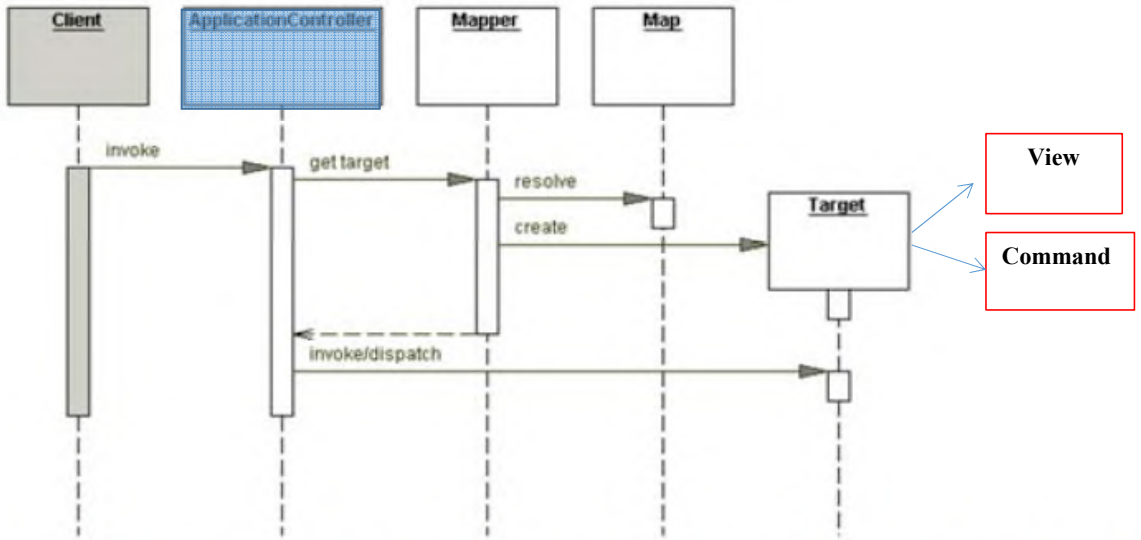
2-Patrón controlador de aplicación

- Estructura



Estructura del patrón controlador de aplicación

2-Patrón controlador de aplicación



Interacción entre objetos relacionados por controlador de aplicación

2- Patrón controlador de aplicación

- Consecuencias
  - Ventajas
    - Mejora la modularidad
    - Mejora la reutilización
    - Mejora la extensibilidad
  - Inconvenientes
    - Aumenta el número de objetos involucrados
    - En aplicaciones grandes puede llegar a crecer mucho

3- Patrón MVC

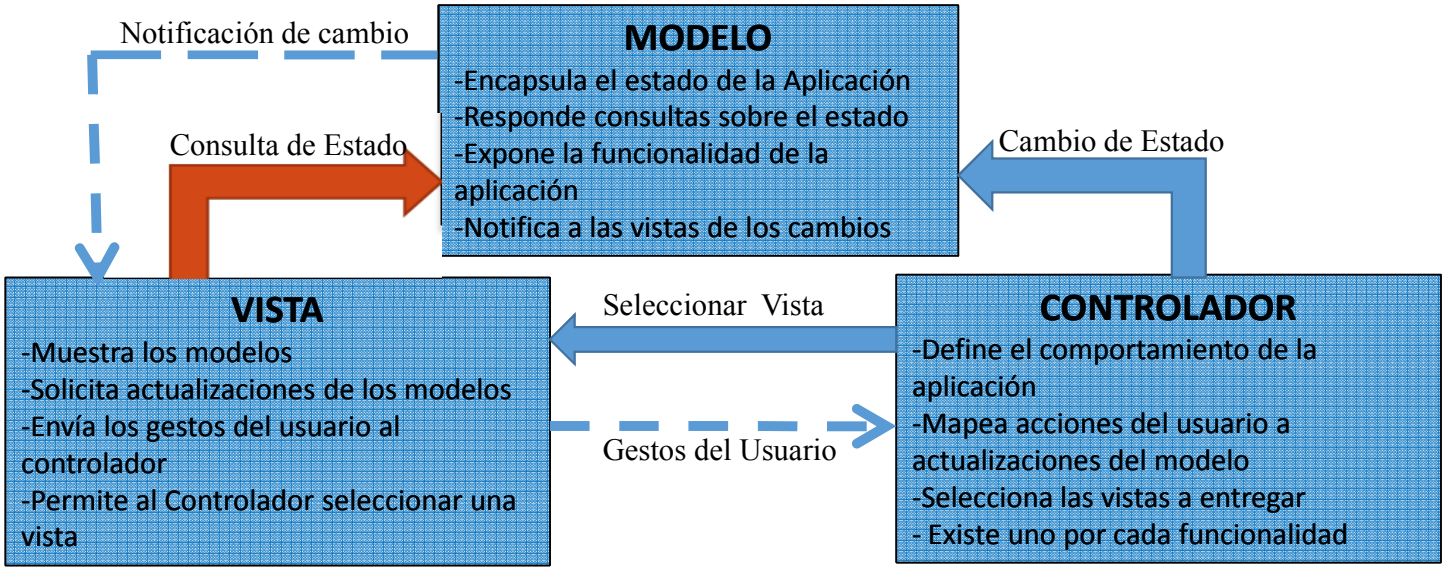
- En muchas aplicaciones no se distingue entre front y application cotroller, y se usa simplemente un controlador que juega ambos roles
- Se habla entonces del patrón *Modelo-Vista-Controlador* (MVC)
- MVC, como tal, no está incluido en el catálogo de patrones multicapa

3-Patrón MVC

- El patrón/arquitectura *Modelo Vista Controlador MVC* divide una aplicación interactiva en tres componentes:

  - El *modelo* contiene la funcionalidad básica y los datos.
  - Las *vistas* muestran/recogen información al/del usuario.
  - Los *controladores* median entre vistas y modelo

3-PATRON MVC

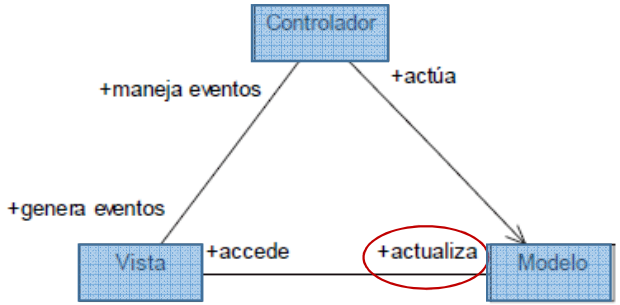

Notificación de cambio
MODELO
-Encapsula el estado de la Aplicación
-Responde consultas sobre el estado
-Expone la funcionalidad de la aplicación
-Notifica a las vistas de los cambios
Consulta de Estado
Cambio de Estado
VISTA
-Muestra los modelos
-Solicita actualizaciones de los modelos
-Envía los gestos del usuario al controlador
-Permite al Controlador seleccionar una vista
Seleccionar Vista
Gestos del Usuario
CONTROLADOR
-Define el comportamiento de la aplicación
-Mapea acciones del usuario a actualizaciones del modelo
-Selecciona las vistas a entregar
- Existe uno por cada funcionalidad

3-Patrón MVC

- El patrón MVC tiene dos variantes:
  - Modelo activo (El Modelo actualiza las vistas)
  - Modelo pasivo (El Controlador actualiza las vistas)

3-Patrón MVC

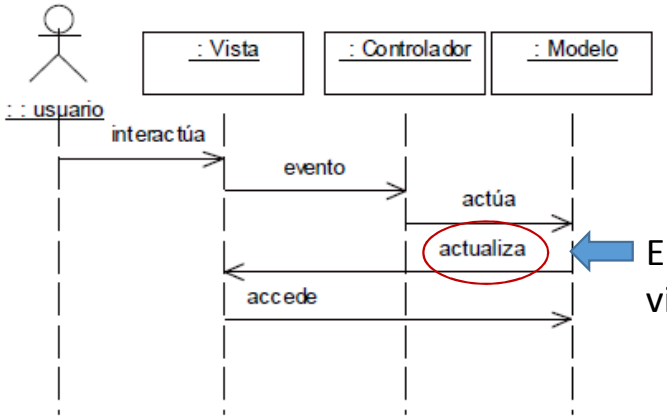
- Participantes en MVC. Modelo activo:



Participantes en MVC. Modelo activo

3-Patrón MVC

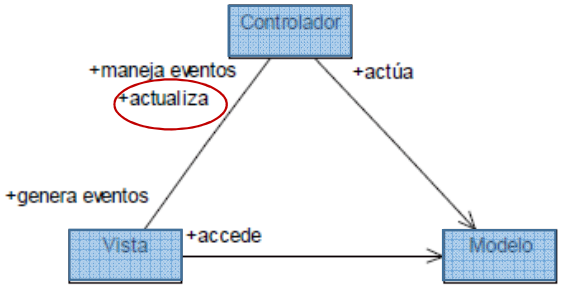
- Interacción en MVC. Modelo activo:



El Modelo actualiza las vistas

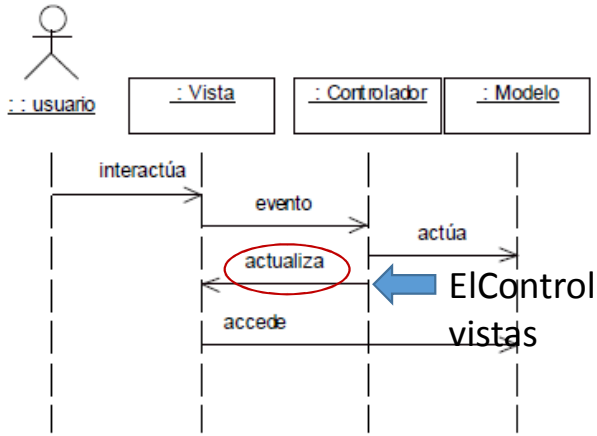
3-Patrón MVC

- Participantes en MVC. Modelo pasivo:



3-Patrón MVC

- Interacción en MVC. Modelo pasivo:



ElControlador actualiza las vistas

3-Patrón MVC

- Ventajas:
  - Modelo independiente de la representación de la salida y del comportamiento de la entrada.
  - Puede haber múltiples vistas para un mismo modelo.
  - Cambios independientes en interfaz/lógica.
- Inconvenientes
  - Complejidad

4- Patrón transferencia

- Propósito
  - Independizar el intercambio de datos entre capas
- También conocido como
  - *Transfer*

4-Patrón transferencia

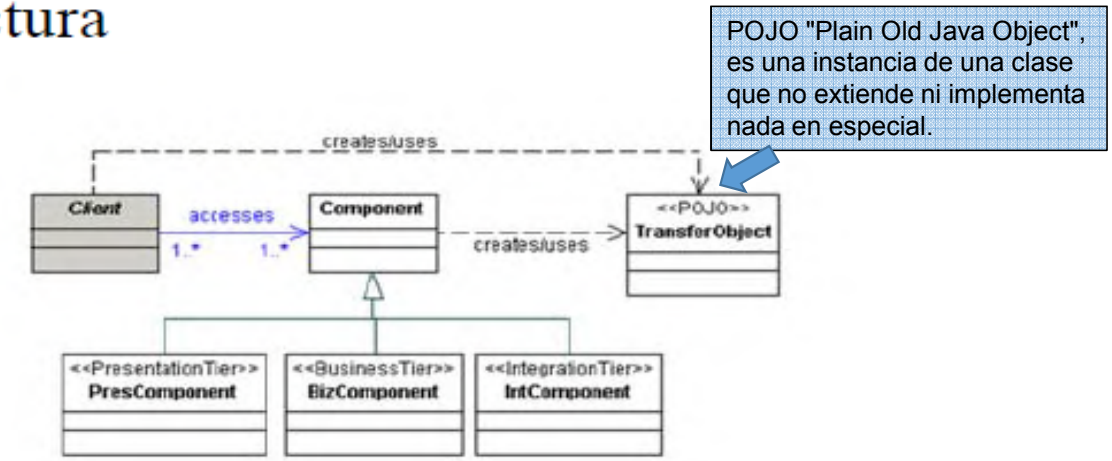
- Motivación
  - Si queremos independizar las capas, éstas no pueden tener conocimiento de la representación de las entidades de nuestro sistema dentro de cada capa
  - Por ejemplo, si accedemos a bases de datos relacionales, los clientes deberían abstraerse de la existencia de *columnas* en los datos

4-Patrón transferencia

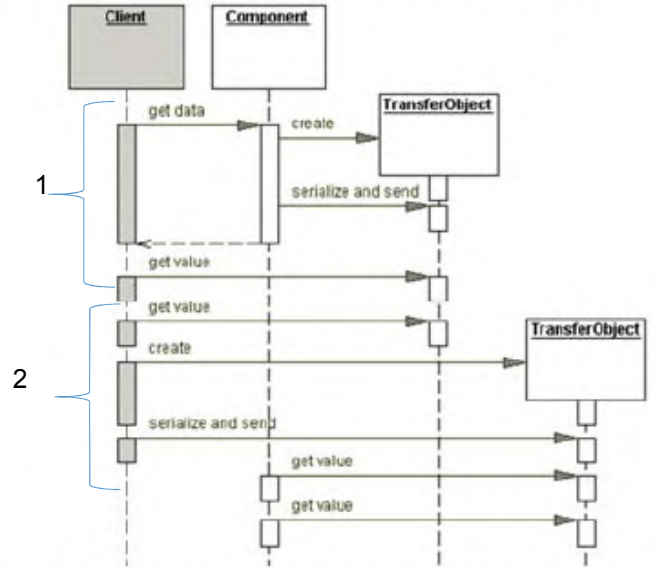
- Debe aplicarse cuando
  - No se desee conocer la representación interna de una entidad dentro de una capa

4-Patrón tranferencia

- Estructura



4-Patrón transferencia



Interacción entre objetos relacionados por el patrón transferencia

4-Patrón transferencia

- Consecuencias
  - Ventajas
    - Ayuda a independizar capas
  - Inconvenientes
    - Aumenta significativamente el número de objetos del sistema

5-Patrón DAO

- Motivación
  - Los sistemas de información (y muchos programas) guardan datos del usuario
  - Estos datos suelen tener estructura, la cual queda plasmada en un sistema de representación (p.e., relacional, XML)

5-Patrón DAO

- Propósito
  - Permite acceder a la capa de datos (recursos, en general), proporcionando representaciones orientadas a objetos (e.g. objetos transferencia) a sus clientes
- También conocido como
  - *Data access object*
  - Objeto de acceso a datos

5-Patrón DAO

- Manejar estos datos fuerza a:
  - Conocer los mecanismos de acceso del sistema de gestión de datos (p.e., base de datos, sistema operativo, etc.)
  - Conocer la representación de los datos en el sistema de gestión de datos (p.e., columnas, elementos, bytes, etc.)
- Un cliente de la capa de negocio debería ser independiente de estas cuestiones

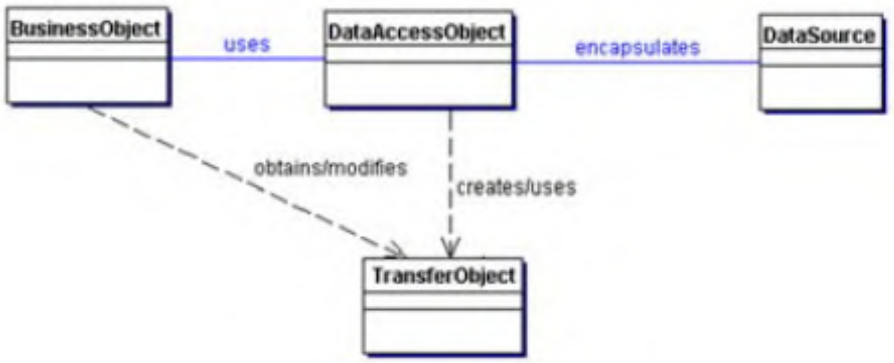
5-Patrón DAO

- Así, se podría cambiar la capa de datos, sin afectar a la capa de negocio. Solamente habría que actualizar la capa de integración, más ligera que la de negocio

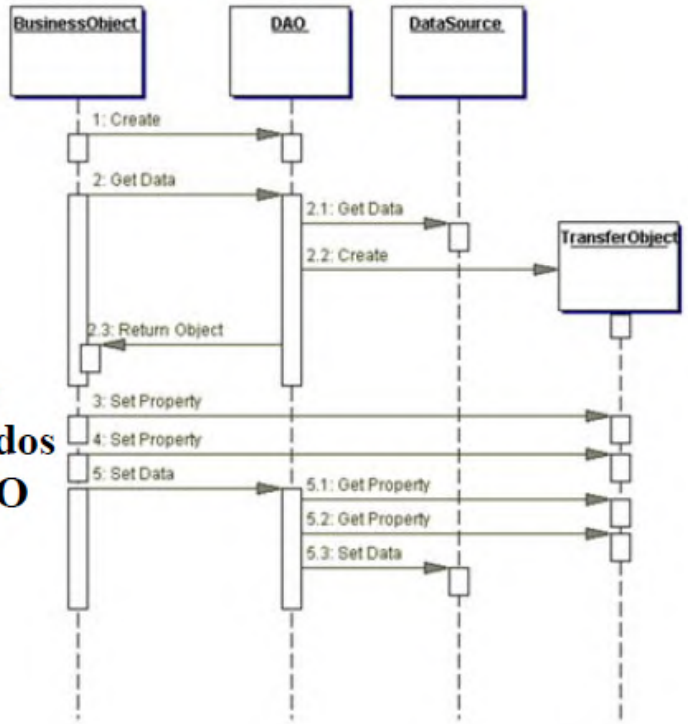
- Debe aplicarse cuando
  - Se quiera independizar la representación y acceso a los datos de su procesamiento

5-Patrón DAO

- Estructura



5-Patrón DAO



Interacción entre objetos relacionados por el patrón DAO

5-Patrón DAO

- Consecuencias
  - Ventajas:
    - Independiza el tratamiento de los datos de su acceso y estructura
    - Permite independizar la capa de negocio de la de datos
  - Inconvenientes
    - Aumenta el número de objetos del sistema

5-Patrón DAO

- fundamental que los DAOs capturen y lancen las excepciones correspondientes al acceder a los recursos externos
- Así, la capa de negocio sabrá qué ha sucedido si ha habido algún tipo de fallo en dicho acceso