Patrones

(Conceptos Generales)

Agenda

- Motivación para los patrones
- Definiciones de patrones
- Tipos de patrones (Catálogos)
  - De diseño (GoF)
  - De arquitectura (POSA)
  - De plataforma (J2EE)
- Ejemplo (Patrón DAO)
  - Contexto y definiciones
  - Ejemplos
- Conclusiones

Agenda

- Motivación para los patrones
- Definiciones de patrones
- Tipos de patrones (Catálogos)
  - De diseño (GoF)
  - De arquitectura (POSA)
  - De plataforma (J2EE)
- Ejemplo (Patrón DAO)
  - Contexto y definiciones
  - Ejemplos
- Conclusiones

Motivación

- Requerimientos funcionales
- Casos de Uso
- Modelamiento del negocio

¿Usted sólo costea esto?

Aplicación empresarial de gran escala = **Lógica Funcional** + **Lógica No funcional**

¿Esto quién lo hace?

- Integración (EAI, Connector)
- Acceso concurrente
- Transacciones distribuidas (ACID)
- Manejo de memoria
- Seguridad.
- Alta disponibilidad
- Manejo de threads.
- Pool de conexiones.
- Escalabilidad

Adaptado de: ARIAS B, Jorge Humberto. Patrones de Software “La lingua-franca de los desarrolladores”. En: XXIII Salón de Informática de ACIS (Sep. 2003: Bogota, Colombia).

Motivación

- Requerimientos funcionales
- Casos de Uso
- Modelamiento del negocio

Aplicación empresarial de gran escala = **Lógica Funcional** + **Lógica No funcional**

J2EE
.NET

- Integración (EAI, Connector)
- Acceso concurrente
- Transacciones distribuidas (ACID)
- Manejo de memoria
- Seguridad.
- Alta disponibilidad
- Manejo de threads.
- Pool de conexiones.
- Escalabilidad

Adaptado de: ARIAS B, Jorge Humberto. Patrones de Software “La lingua-franca de los desarrolladores”. En: XXIII Salón de Informática de ACIS (Sep. 2003: Bogota, Colombia).

Motivación

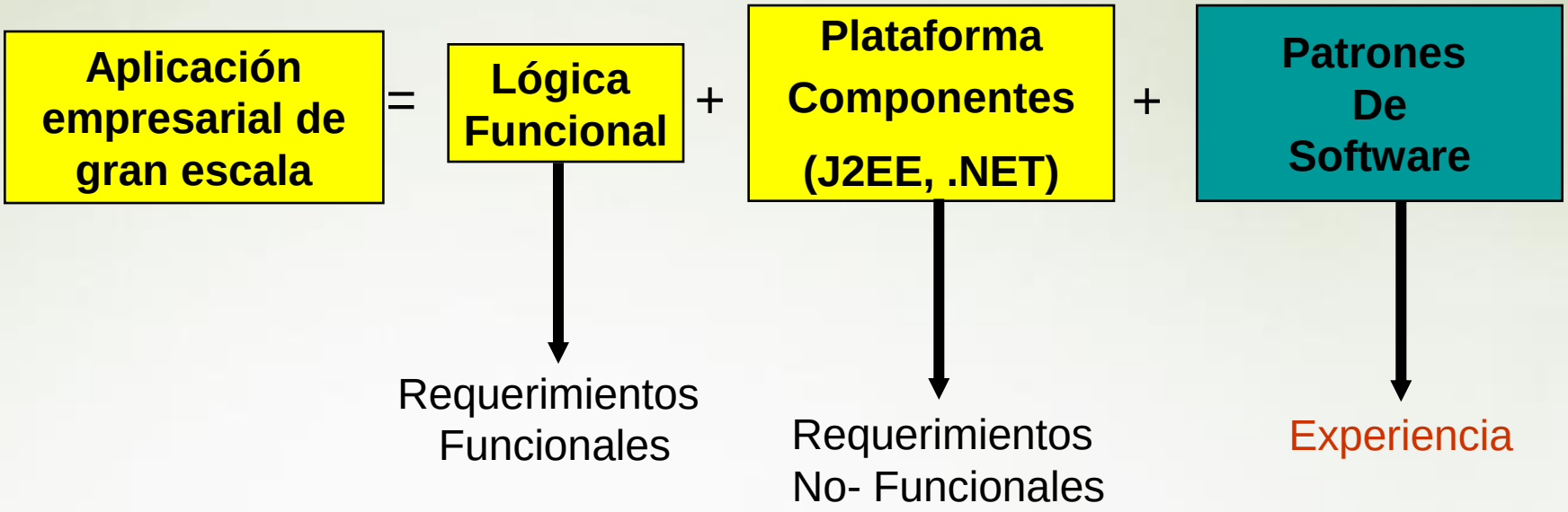
Libros
- Aprendiendo J2EE en 21 días.
- J2EE al instante.

¿ **Aplicación empresarial de gran escala** = Requerimientos Funcionales **Lógica Funcional** + **Plataforma Componentes (J2EE, .NET)** Requerimientos No- Funcionales ?

¿Será suficiente?

Adaptado de: ARIAS B, Jorge Humberto. Patrones de Software “La lingua-franca de los desarrolladores”. En: XXIII Salón de Informática de ACIS (Sep. 2003: Bogota, Colombia).

Motivación



"¿Será la formula del éxito?"

Fue formula del éxito !!!!

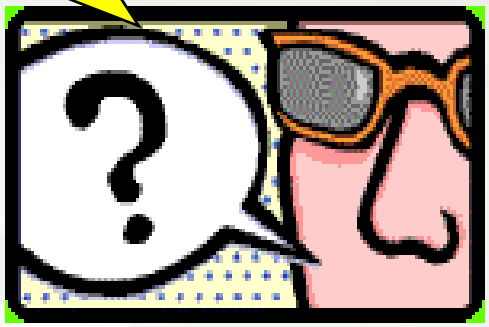
Adaptado de: ARIAS B, Jorge Humberto. Patrones de Software "La lingua-franca de los desarrolladores". En: XXIII Salón de Informática de ACIS (Sep. 2003: Bogota, Colombia).

Motivación

¿ Cómo hago para eliminar el tiempo de conexión a la base de datos en cada invocación http, y tener así tiempos de respuesta más bajos?

Consultor en arquitectura y plataformas

FACIL!!, Haga un singleton, que contenga un resource pooling que adapte a clases tipo abstract factory de conexiones, e implemente un observer para notificar el estado de la conexión.

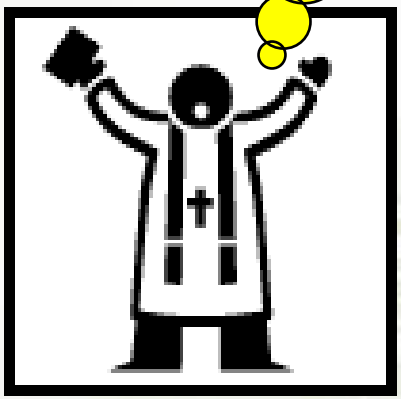


Desarrollador

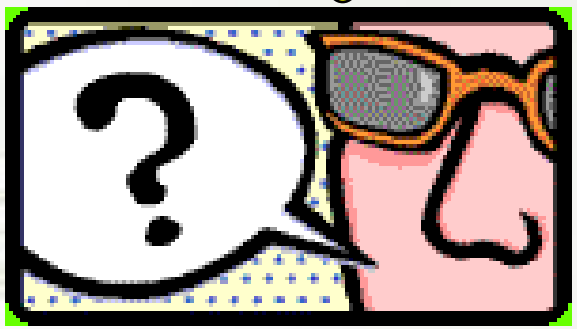
Adaptado de: ARIAS B, Jorge Humberto. Patrones de Software “La lingua-franca de los desarrolladores”. En: XXIII Salón de Informática de ACIS (Sep. 2003: Bogota, Colombia).

Motivación

¿Por qué estará tan callado? ¿Será que no está de acuerdo conmigo en que sea un abstract factory?



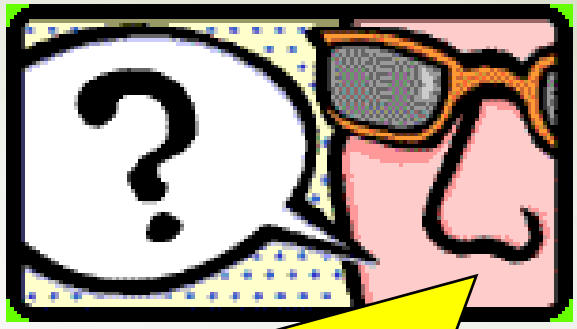
Este tipo está loco!!! No le he entendido nada. ¿De qué me hablaría? ¿En qué idioma me hablaría?



Adaptado de: ARIAS B, Jorge Humberto. Patrones de Software “La lingua-franca de los desarrolladores”. En: XXIII Salón de Informática de ACIS (Sep. 2003: Bogota, Colombia).

Motivación

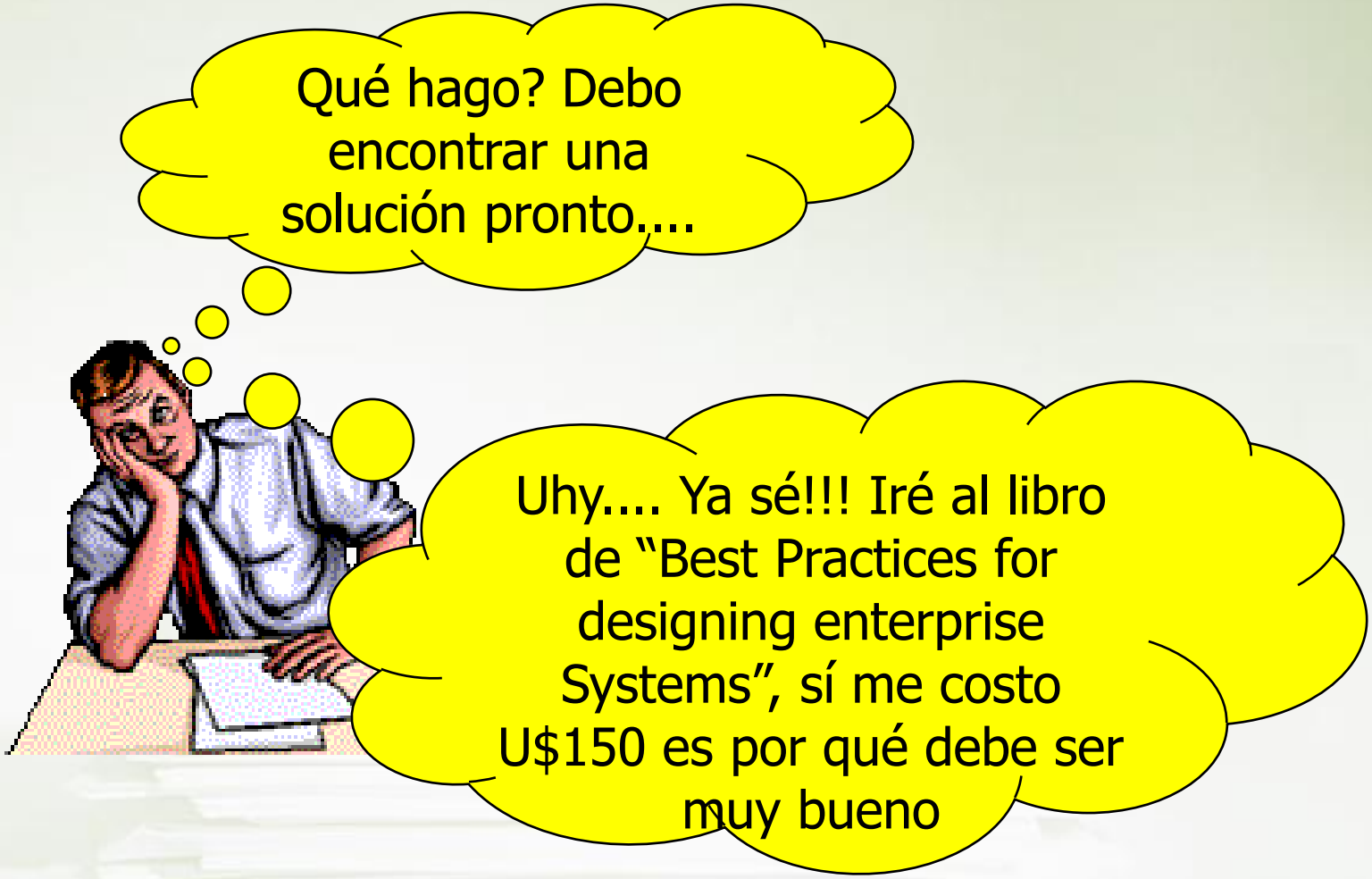
Sí no estás de acuerdo con el abstract factory, no hay problema, con el method factory también funciona



No, no, por el contrario creo que la mejor opción es usar un abstract de esos... Realmente, me ha quedado muy clara la solución... Tu siempre tan claro!!!

Adaptado de: ARIAS B, Jorge Humberto. Patrones de Software “La lingua-franca de los desarrolladores”. En: XXIII Salón de Informática de ACIS (Sep. 2003: Bogota, Colombia).

Motivación

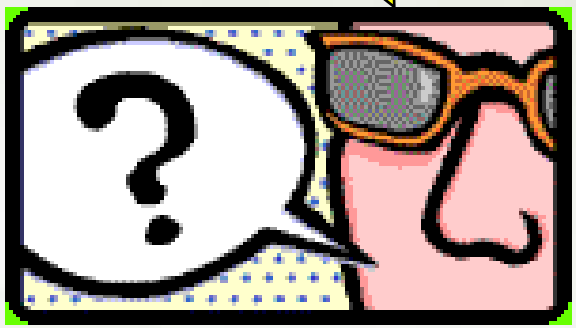


Adaptado de: ARIAS B, Jorge Humberto. Patrones de Software "La lingua-franca de los desarrolladores". En: XXIII Salón de Informática de ACIS (Sep. 2003: Bogota, Colombia).

Motivación

Best practices for designing enterprise systems

(Buscando....)
"Elimar el tiempo de conexión a la base de datos en cada invocación http"



Implemente un resource pooling singleton que adapte clases abstract factories, y se apoye en un modelo de notificación tipo call-back, tal como un observer remoto.

Adaptado de: ARIAS B, Jorge Humberto. Patrones de Software “La lingua-franca de los desarrolladores”. En: XXIII Salón de Informática de ACIS (Sep. 2003: Bogota, Colombia).

Motivación

- Para que solucionar problemas que otros han resuelto?
- Si alguien ya lo resolvió, Cómo comunicar experiencias? Cómo comunicar diseños?
  - Todos tenemos ideas diferentes de un mismo concepto.
- Emplear un lenguaje que sea comprensible por los desarrolladores, diseñadores, arquitectos.

Motivación

Los patrones de software permiten establecer un lenguaje común para expresar y comunicar experiencias, diseños y buenas prácticas.

Agenda

- Motivación para los patrones
- Definiciones de patrones
- Tipos de patrones (Catálogos)
  - De diseño (GoF)
  - De arquitectura (POSA)
  - De plataforma (J2EE)
- Ejemplo (Patrón DAO)
  - Contexto y definiciones
  - Ejemplos
- Conclusiones

Definición

- Qué es un patrón?

“Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo siquiera dos veces de la misma forma”.

[Christopher Alexander]

Definición

- Qué es un patrón de software?

“Un patrón describe un problema de diseño recurrente, que surge en contextos específicos de diseño, y presenta un esquema genérico probado para la solución de este.
El esquema de la solución describe un conjunto de componentes, responsabilidades y relaciones entre de éstos, y formas en que dichos componentes colaboran entre sí.”

[Buschmann]

Definición

- “Un patrón describe un problema de diseño recurrente,
- que surge en contextos específicos de diseño,
- y presenta un esquema genérico probado para la solución de este.
- El esquema de la solución describe un conjunto de componentes, responsabilidades y relaciones entre de éstos,
- y formas en que dichos componentes colaboran entre sí.”

Patrón Documentario (I)

- Intención: sucinta descripción de lo que se pretende conseguir con el patrón.
- También Conocido como: otros nombres del mismo patrón.
- Motivo: explicación justificativa de la necesidad de que el patrón exista como entidad autónoma.
- Aplicabilidad: lista de usos para los que resulta especialmente adecuado el patrón que se describe.
- Estructura: descripción gráfica de los comportamientos, acciones y relaciones de los objetos que participan en el patrón.

Patrón Documentario (II)

- Participantes: diccionario de las partes que componen el patrón.
- Colaboraciones: diccionario de las relaciones e interacciones entre los participantes en un patrón.
- Consecuencias: detalle de los posibles beneficios y perjuicios que pueden derivarse del uso del patrón.
- Implementación: detalle de las posibles implementaciones y catálogo de las decisiones de diseño en la codificación de soluciones concretas basadas en el patrón.

Patrón Documentario (III)

- Código de Ejemplo: planteamiento de código práctico referido a un ejemplo (o ejemplos) suficientemente representativo del uso del patrón.
- Usos Conocidos: detalle de bibliotecas, productos y sistemas en que se ha utilizado el patrón.
- Patrones Relacionados: referencias a otros patrones que bien son directamente utilizados por el descrito bien representan soluciones complementarias o suplementarias al mismo.

Características

- No son inventados, provienen de la experiencia practica.
- Describen un grupo de componentes, sus relaciones, interacciones y responsabilidades.
- Proveen un vocabulario común entre los diseñadores, desarrolladores.
- Ayudan a documentar la visión de arquitectura de un diseño.
- Proveen una plantilla / molde conceptual para dar solución a un problema de diseño.

Agenda

- Motivación para los patrones
- Definiciones de patrones
- Tipos de patrones (Catálogos)
  - De diseño (GoF)
  - De arquitectura (POSA)
  - De plataforma (J2EE)
- Ejemplo (Patrón DAO)
  - Contexto y definiciones
  - Ejemplos
- Conclusiones

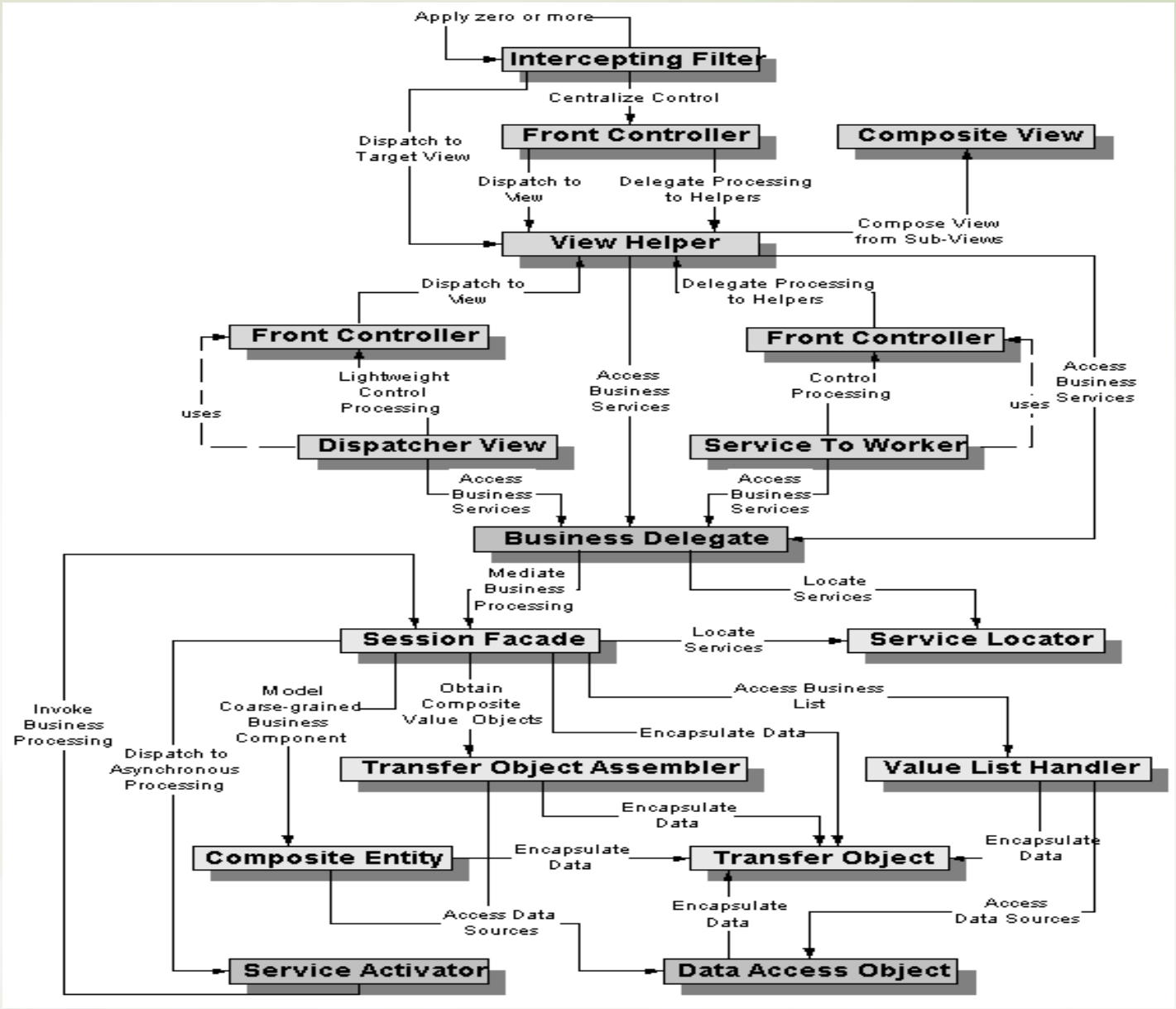
Catálogo de GoF

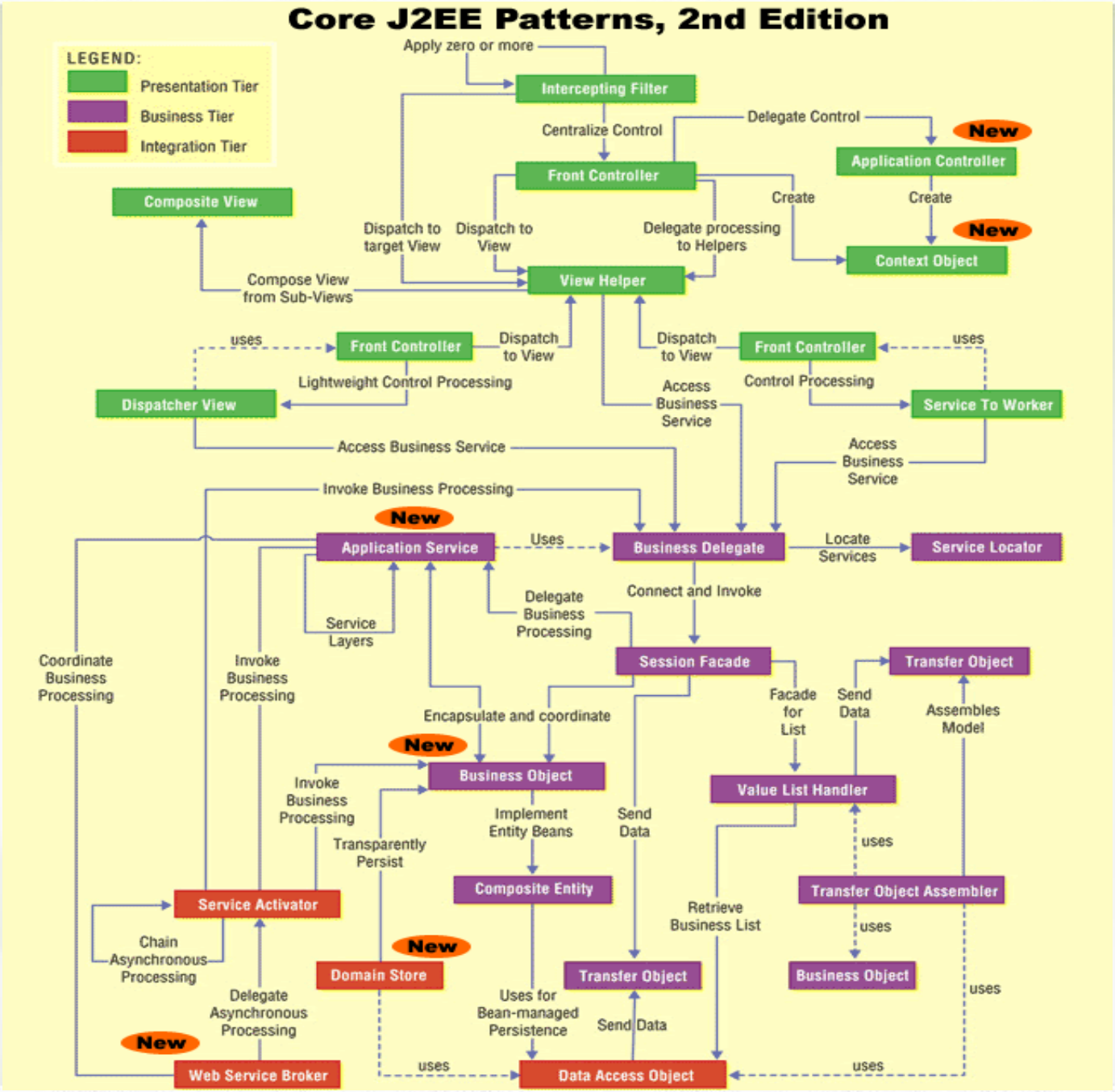
		Propósito		
		Creación	Estructural	Comportamiento
Ámbito	Clase	Factory Method	Adapter	Interpreter Template Method
	Objeto	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Catálogo POSA

Del fango a la estructura	- Capas - Tubería-filtros - Pizarra
Sistemas distribuidos	- Broker (p. ej. CORBA, DCOM, Web Services, WWW)
Sistemas interactivos	- Model-View-Controller - Presentation-Abstraction-Control
Sistemas adaptables	- Reflection: metanivel que hace al software consciente de sí mismo. - Microkernel: núcleo de funcionalidad mínima.

Catálogo J2EE (1° Edición)




Core J2EE Patterns, 2nd Edition
LEGEND:
Presentation Tier
Business Tier
Integration Tier
Apply zero or more
Intercepting Filter
Centralize Control
Delegate Control
New
Application Controller
Front Controller
Create
Create
New
Context Object
Composite View
Dispatch to target View
Dispatch to View
Delegate processing to Helpers
View Helper
Compose View from Sub-Views
uses
Front Controller
Dispatch to View
Dispatch to View
Front Controller
uses
Lightweight Control Processing
Control Processing
Dispatcher View
Access Business Service
Service To Worker
Access Business Service
Access Business Service
Invoke Business Processing
New
Application Service
Uses
Business Delegate
Locate Services
Service Locator
Service Layers
Delegate Business Processing
Connect and Invoke
Coordinate Business Processing
Invoke Business Processing
Session Facade
Transfer Object
Facade for List
Send Data
Assembles Model
Encapsulate and coordinate
New
Business Object
Value List Handler
Invoke Business Processing
Implement Entity Beans
Send Data
Transparently Persist
uses
Composite Entity
Transfer Object Assembler
Retrieve Business List
Service Activator
uses
Chain Asynchronous Processing
New
Domain Store
Transfer Object
Business Object
uses
Delegate Asynchronous Processing
Uses for Bean-managed Persistence
Send Data
New
Web Service Broker
uses
Data Access Object
uses

J2EE - Capa de Presentación

Decorating Filter / Intercepting Filter	Un objeto que está entre el cliente y los componentes Web. Este procesa las peticiones y las respuestas.
Front Controller/ Front Component	Un objeto que acepta todos los requerimientos de un cliente y los direcciona a manejadores apropiados. El patrón Front Controller podría dividir la funcionalidad en 2 diferentes objetos: el Front Controller y el Dispatcher. En ese caso, El Front Controller acepta todos los requerimientos de un cliente y realiza la autenticación, y el Dispatcher direcciona los requerimientos a manejadores apropiada.
View Helper	Un objeto helper que encapsula la lógica de acceso a datos en beneficio de los componentes de la presentación. Por ejemplo, los JavaBeans pueden ser usados como patrón View Helper para las páginas JSP.
Composite view	Un objeto vista que está compuesto de otros objetos vista. Por ejemplo, una página JSP que incluye otras páginas JSP y HTML usando la directiva include o el action include es un patrón Composite View.
Service To Worker	Es como el patrón de diseño MVC con el Controlador actuando como Front Controller pero con una cosa importante: aquí el Dispatcher (el cual es parte del Front Controller) usa View Helpers a gran escala y ayuda en el manejo de la vista.
Dispatcher View	Es como el patrón de diseño MVC con el controlador actuando como Front Controller pero con un asunto importante: aquí el Dispatcher (el cual es parte del Front Controller) no usa View Helpers y realiza muy poco trabajo en el manejo de la vista. El manejo de la vista es manejado por los mismos componentes de la Vista.

J2EE - Capa de Negocio

Business Delegate	Un objeto que reside en la capa de presentación y en beneficio de los otros componentes de la capa de presentación llama a métodos remotos en los objetos de la capa de negocios.
Value Object/ Data Transfer Object/ Replicate Object	Un objeto serializable para la transferencia de datos sobre lar red.
Session Façade/ Session Entity Façade/ Distributed Façade	El uso de un bean de sesion como una fachada (facade) para encapsular la complejidad de las interacciones entre los objetos de negocio y participantes en un flujo de trabajo. El Session Façade maneja los objetos de negocio y proporciona un servicio de acceso uniforme a los clientes.
Aggregate Entity	Un bean entidad que es construido o es agregado a otros beans de entidad.
Value Object Assembler	Un objeto que reside en la capa de negocios y crea Value Objets cuando es requerido.
Value List Handler/ Page-by-Page Iterator/ Paged List	Es un objeto que maneja la ejecución de consultas SQL, caché y procesamiento del resultado. Usualmente implementado como beans de sesión.
Service Locator	Consiste en utilizar un objeto Service Locutor para abstraer toda la utilización JNDI y para ocultar las complejidades de la creación del contexto inicial, de búsqueda de objetos home EJB y recreación de objetos EJB. Varios clientes pueden reutilizar el objeto Service Locutor para reducir la complejidad del código, proporcionando un punto de control.

J2EE - Capa de Integración

Data Access Object

Consiste en utilizar un objeto de acceso a datos 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.

Service Activator

Se utiliza para recibir peticiones y mensajes asíncronos de los clientes. Cuando se recibe un mensaje, el Service Activator localiza e invoca a los métodos de los componentes de negocio necesarios para cumplir la petición de forma asíncrona.

Agenda

- Motivación para los patrones
- Definiciones de patrones
- Tipos de patrones (Catálogos)
  - De diseño (GoF)
  - De arquitectura (POSA)
  - De plataforma (J2EE)
- Ejemplo (Patrón DAO)
  - Contexto y definiciones
  - Ejemplos
- Conclusiones

J2EE - Patrón DAO

- J2EE (Java 2 Enterprise Edition) es una plataforma para el desarrollo de aplicaciones empresariales distribuidas.

- Patrones de la capa de presentación
- Patrones de la capa de negocios
- **Patrones de la capa de integración**

Data Access Object (DAO)

Intención:

- Desacoplar la lógica de negocio de la lógica de acceso a datos, de manera que se pueda cambiar la fuente de datos fácilmente

DAO

Problemas:

- El acceso a los datos varía dependiendo de la fuente de los datos.
- Mezclar la lógica de persistencia con la lógica del negocio crea dependencia.

DAO

¿Cuando usarlo?

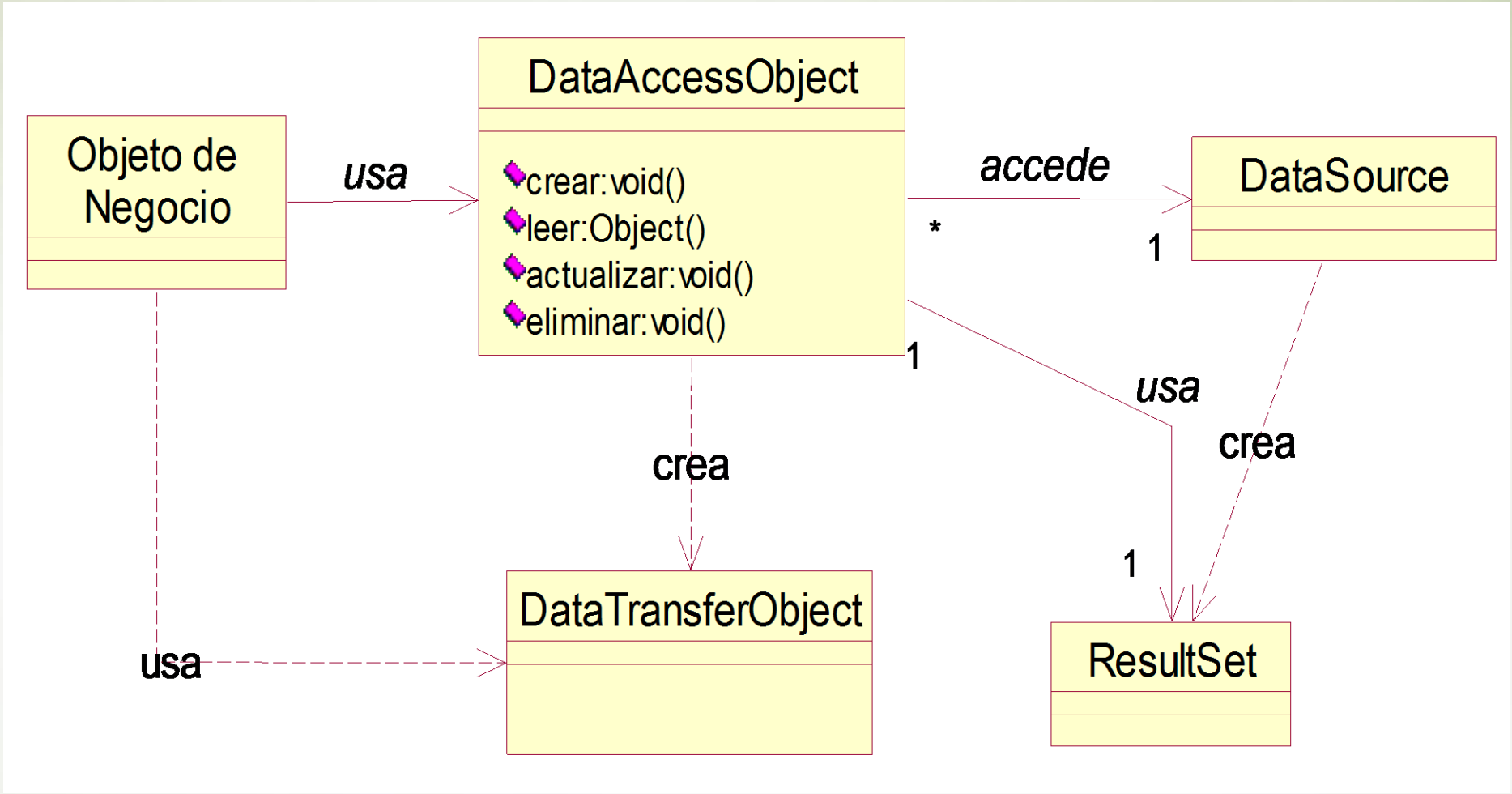
- Deseas implementar un mecanismo para acceder y manipular la data en un repositorio persistente.
- Deseas desacoplar aplicación - repositorio.
- Deseas proveer un API uniforme de acceso a datos que soporte varios tipos de repositorios.
- Deseas organizar la lógica de acceso a datos para facilitar el mantenimiento y portabilidad.

DAO

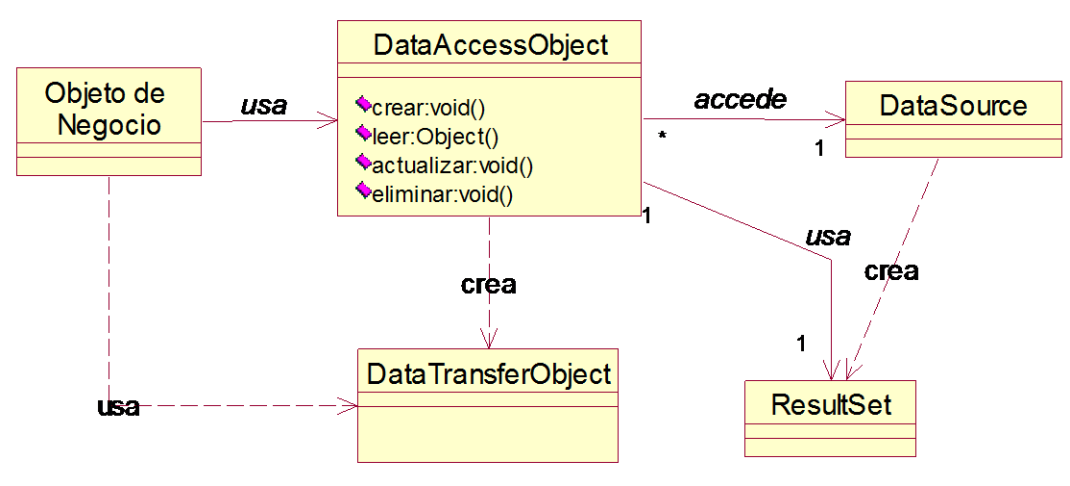
Solución

- Utilizar un Data Access Object (DAO) para abstraer y encapsular todos los accesos a la fuente de datos
- No debe contener ningún método de negocio.
- Permitir que la implementación de la persistencia sea fácilmente remplazada por otra, sin afectar negativamente a los objetos de negocio.

DAO (Estructura)

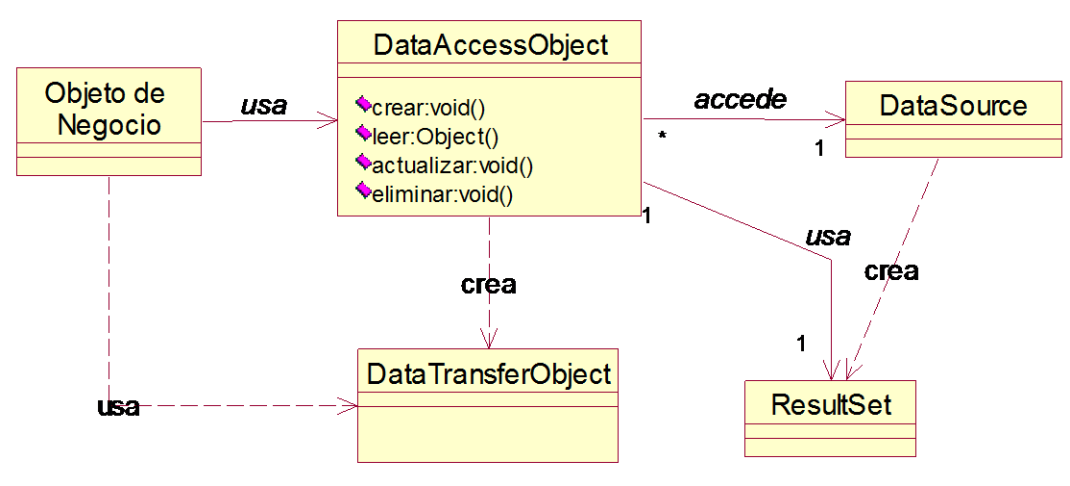


Responsabilidades



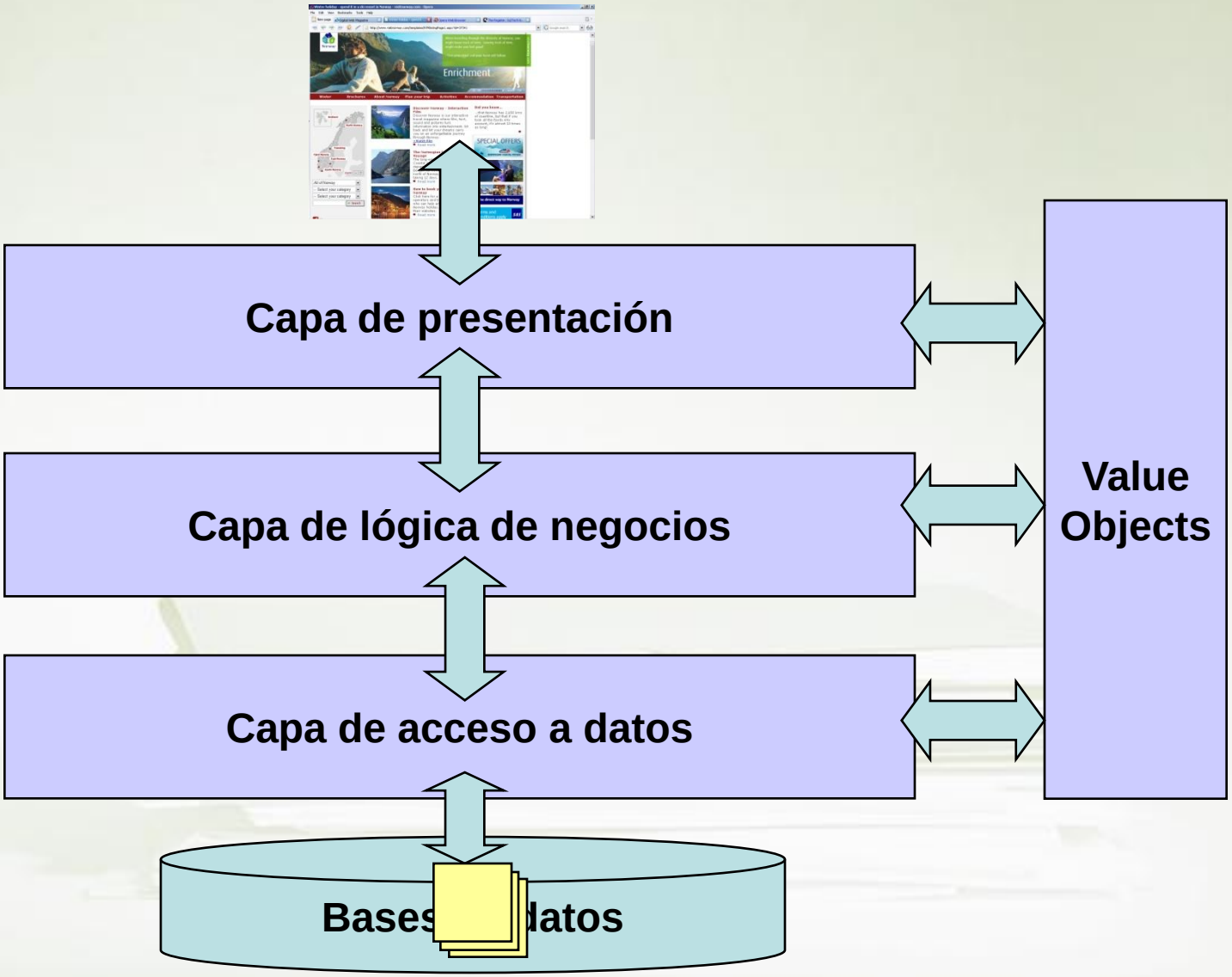
- **Objeto de Negocio:** Cualquier objeto que requiera acceder a la fuente de datos para obtener algún dato.
- **DataAccessObject:** Implementación de la las operaciones de acceso a datos (CRUD o CRUDEL)

Responsabilidades



- **DataSource:** Implementación de la fuente de datos
- **ResultSet:** Interfaz ResultSet proporciona acceso a una tabla de datos.
- **Data Transfer Object:** Almacena los datos que deseemos insertar a una tabla y almacena los datos que saquemos de una tabla.

El Papel del DTO o Value Object (J2EE)



Conclusiones - DAO

- Objeto de Negocio
- Permite la Transparencia
- Permite una Migración más Fácil
- Reduce la Complejidad del Código de los Objetos de Negocio
- Centraliza Todos los Accesos a Datos en un Capa Independiente
- Añade una Capa Extra (-)

Agenda

- Motivación para los patrones
- Definiciones de patrones
- Tipos de patrones (Catálogos)
  - De diseño (GoF)
  - De arquitectura (POSA)
  - De plataforma (J2EE)
- Ejemplo (Patrón DAO)
  - Contexto y definiciones
  - Ejemplos
- Conclusiones

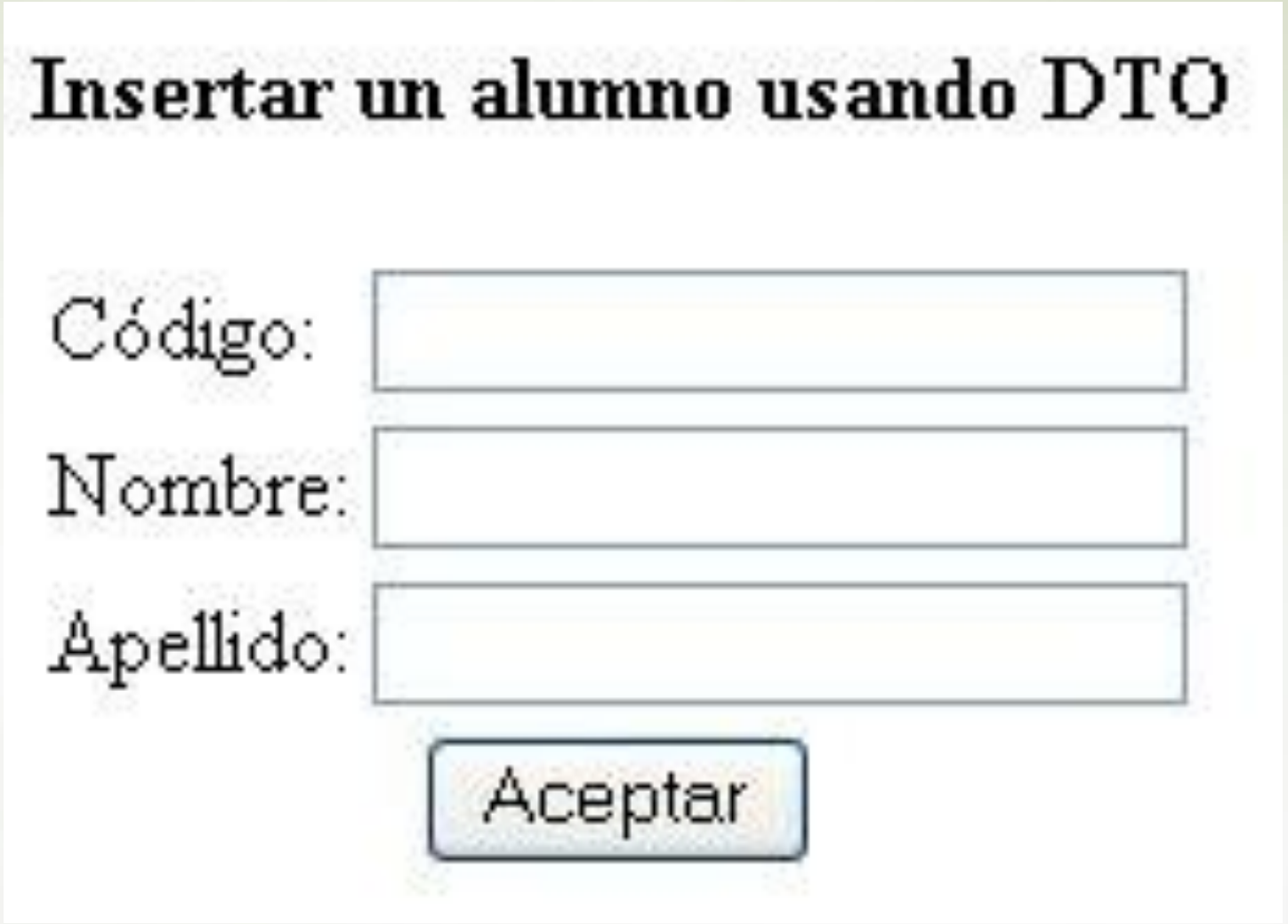
Flujos en el patrón DAO

- **Flujos de un registro**: manipulan un solo registro de la fuente de datos y no necesitan ResultSet.

- **Flujos de múltiples registros**: manipulan mas de un registro de la fuente de datos usando para ello un ResultSet.

** En el patrón **MVC** los flujos se separan en:
  - Flujos para actualizar información
  - Flujos para mostrar información

Ejemplo de un registro



Crear el DTO y pasarle los datos

```
public class AlumnoAction extends org.apache.struts.action.Action
{
    public ActionForward execute(ActionMapping mapping, ActionForm
    {
        AlumnoForm alumnoForm = (AlumnoForm) form;

        AlumnoDTO  dto = new AlumnoDTO();
        BeanUtils.copyProperties(dto, alumnoForm);

        AlumnoDAO  dao = new AlumnoDAO();
        dao.insertarAlumno(dto);

        return mapping.findForward("exito");
    }
}
```

Guardar los datos en el DTO

```
public class AlumnoDTO
{
    private String codigo, nombre, apellido;

    public void setCodigo(String codigo) {
        this.codigo = codigo;
    }

    public String getCodigo() {
        return codigo;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getNombre() {
        return nombre;
    }
```

Crear el DAO y pasarle el DTO

```
public class AlumnoAction extends org.apache.struts.action.Action
{
    public ActionForward execute(ActionMapping mapping, ActionForm
    {
        AlumnoForm alumnoForm = (AlumnoForm) form;

        AlumnoDTO  dto = new AlumnoDTO();
        BeanUtils.copyProperties(dto, alumnoForm);

        AlumnoDAO  dao = new AlumnoDAO();
        dao.insertarAlumno(dto);

        return mapping.findForward("exito");
    }
}
```

Ejecutar el método insertarAlumno

```
public class AlumnoDAO
{
    Conexion ds = new Conexion();

    public void insertarAlumno(AlumnoDTO dto) {

        Connection  cn = ds.getConexion();
        Statement   st = null;

        String sql = "INSERT INTO Alumno " +
                     "VALUES('" + dto.getCodigo() + "', " +
                            "'" + dto.getNombre() + "', " +
                            "'" + dto.getApellido() + "') ";
        try
        {
            st = cn.createStatement();
            st.execute(sql);
        }
        catch(Exception e) {}
```

Clase Conexion

```
public class Conexion
{
    private String  driver= "sun.jdbc.odbc.JdbcOdbcDriver";
    private String  url= "jdbc:odbc:dsnDAO";
    private String  login= "sa";
    private String  password= "";

    public Connection getConexion()
    {
        Connection  cn=null;

        try
        {
            Class.forName(driver);
            cn= DriverManager.getConnection(url, login, password);
        }
        catch(SQLException e){}
        catch(Exception e){}

        return cn;
    }
}
```

Diagrama de Clases

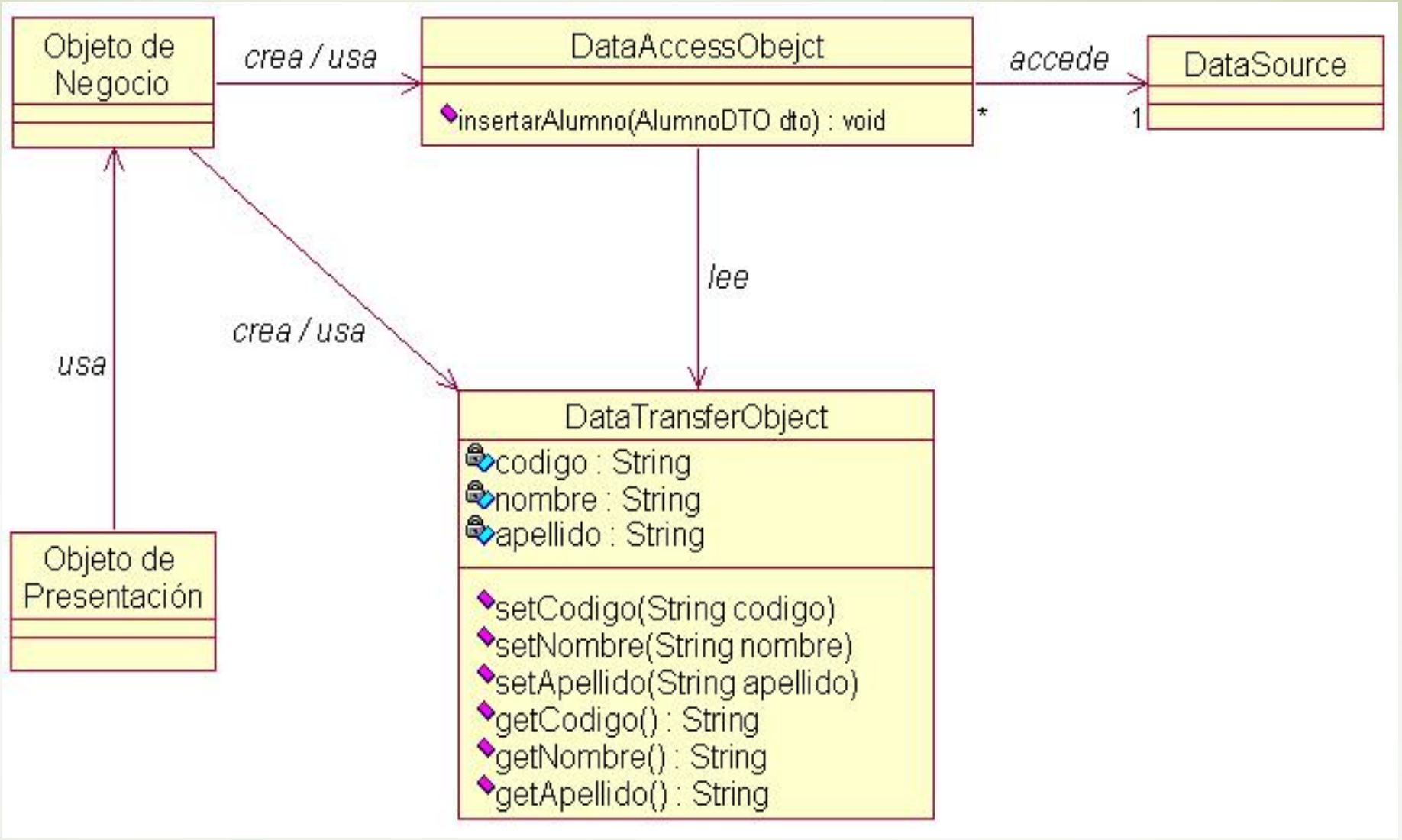


Diagrama de Secuencias

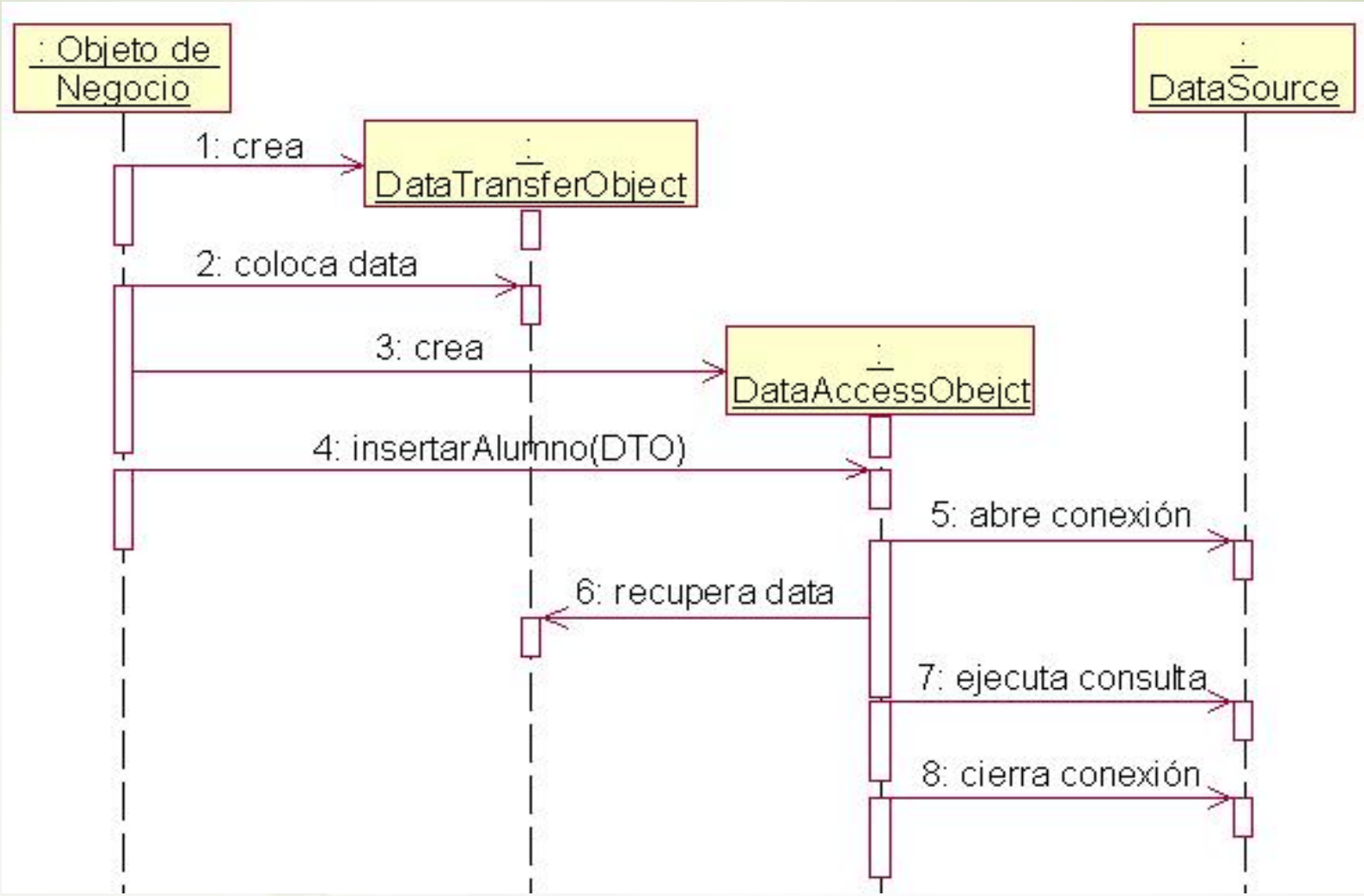
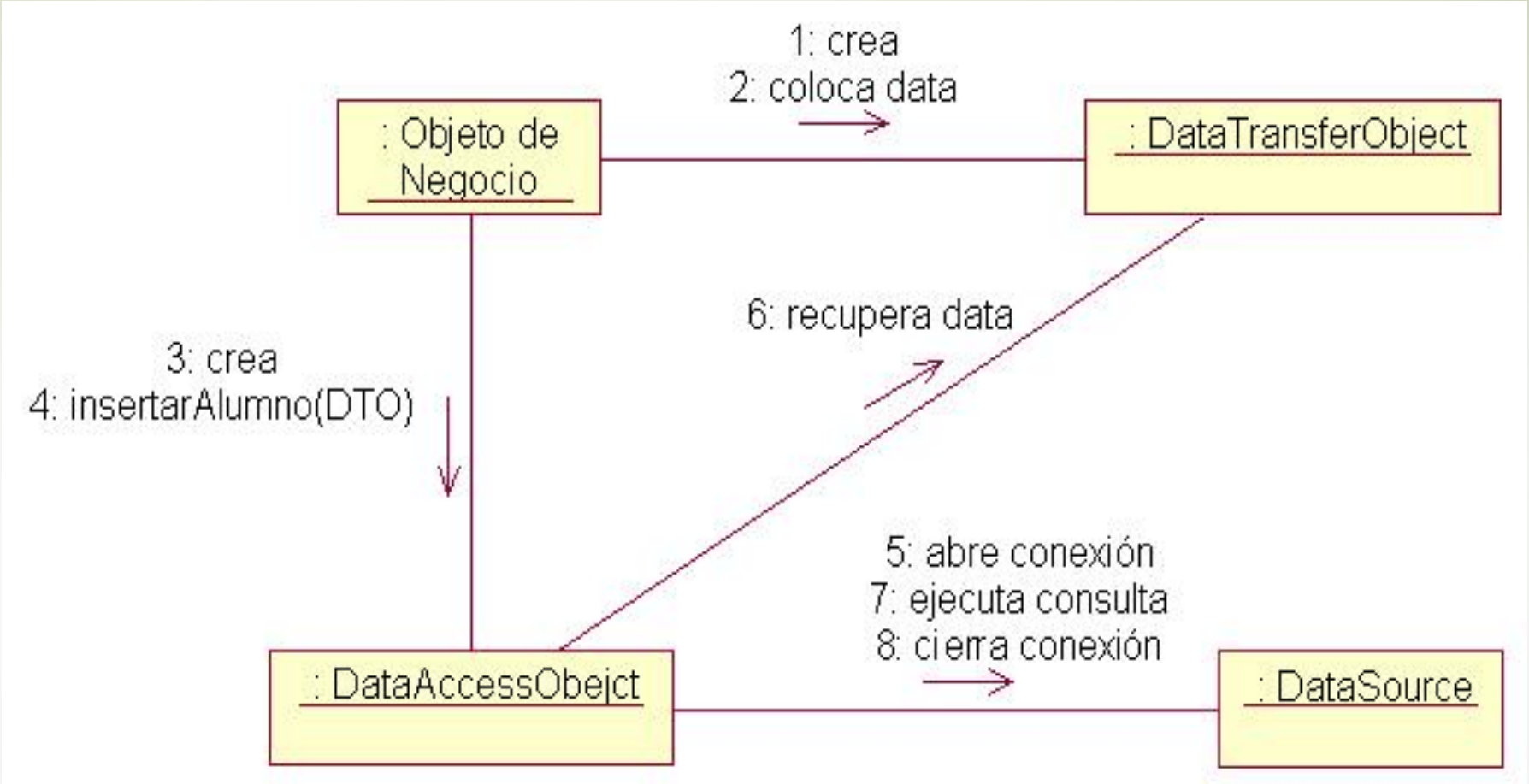


Diagrama de Colaboración



Ejemplo de múltiples registros

Mostrar todos los alumnos usando colección de DTO

Buscar Alumnos

Crear el DAO y llamar al método

```
public class AlumnoAction extends Action
{
    public ActionForward execute(ActionMapping mapping,
    {
        AlumnoDAO dao = new AlumnoDAO();
        List dto = dao.buscarAlumnos();
        request.setAttribute("alumnos", dto);
        return mapping.findForward("exito");
    }
}
```

Ejecuta el método buscarAlumno

```
public class AlumnoDAO
{
    Conexion ds = new Conexion();

    public List buscarAlumnos()
    {
        AlumnoDTO alumno = null;

        Connection  cn = null;
        Statement   st = null;
        ResultSet   rs = null;
        ArrayList   alumnosList = null;
```

Ejecuta el método buscarAlumno

```
try
{
    cn = ds.getConexion();
    st = cn.createStatement();
    rs = st.executeQuery("SELECT * FROM Alumno");

    alumnosList = new ArrayList();

    while (rs.next())
    {
        alumno = new AlumnoDTO();
        alumno.setCodigo(rs.getString(1));
        alumno.setNombre(rs.getString(2));
        alumno.setApellido(rs.getString(3));

        alumnosList.add(alumno);
    }
    return alumnosList;
}
```

Ejecuta el método buscarAlumno

```
public class AlumnoAction extends Action
{
    public ActionForward execute(ActionMapping mapping,
    {
        AlumnoDAO dao = new AlumnoDAO();
        List dto = dao.buscarAlumnos();
        request.setAttribute("alumnos", dto);
        return mapping.findForward("exito");
    }
}
```

Diagrama de Clases

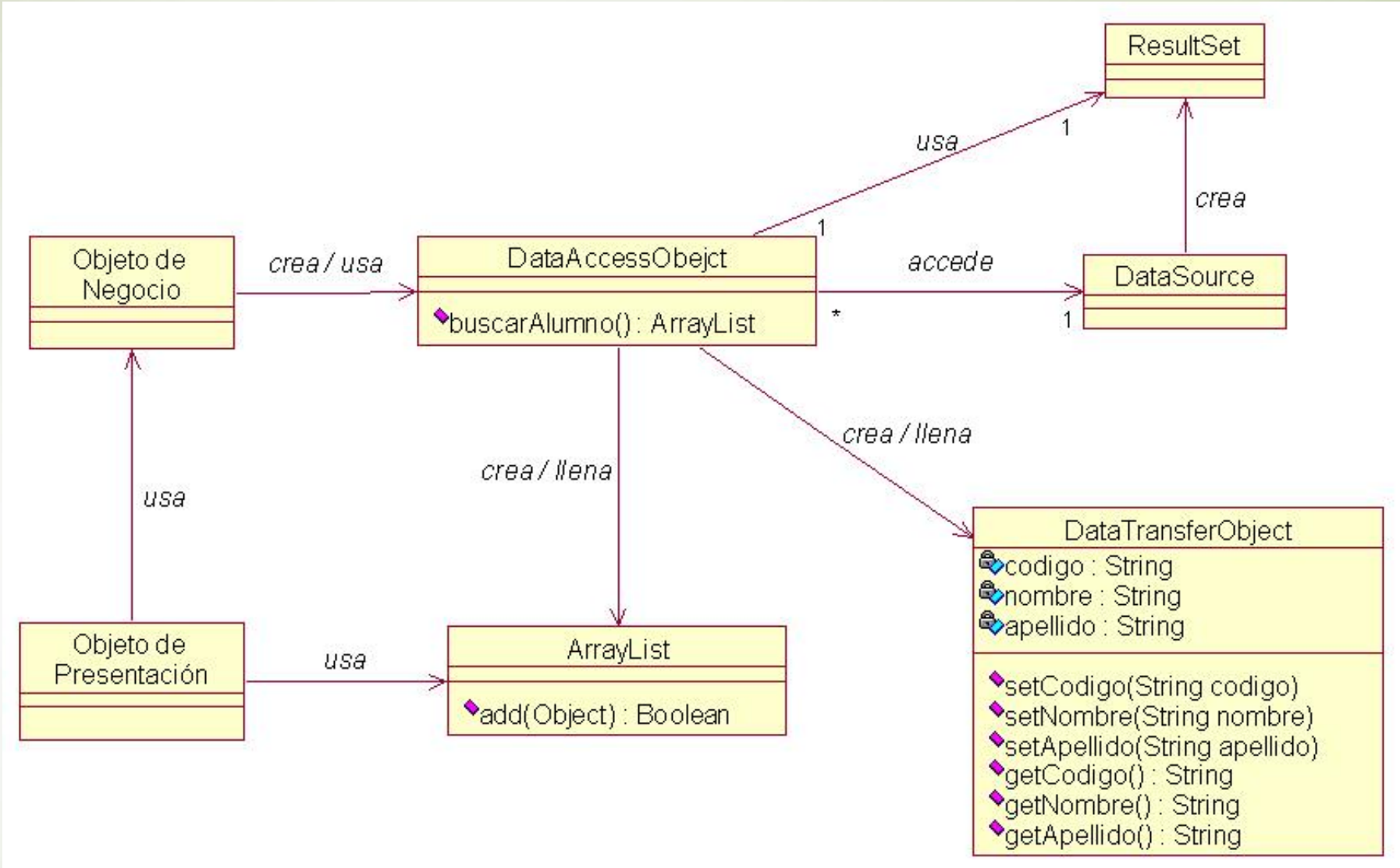


Diagrama de Secuencias

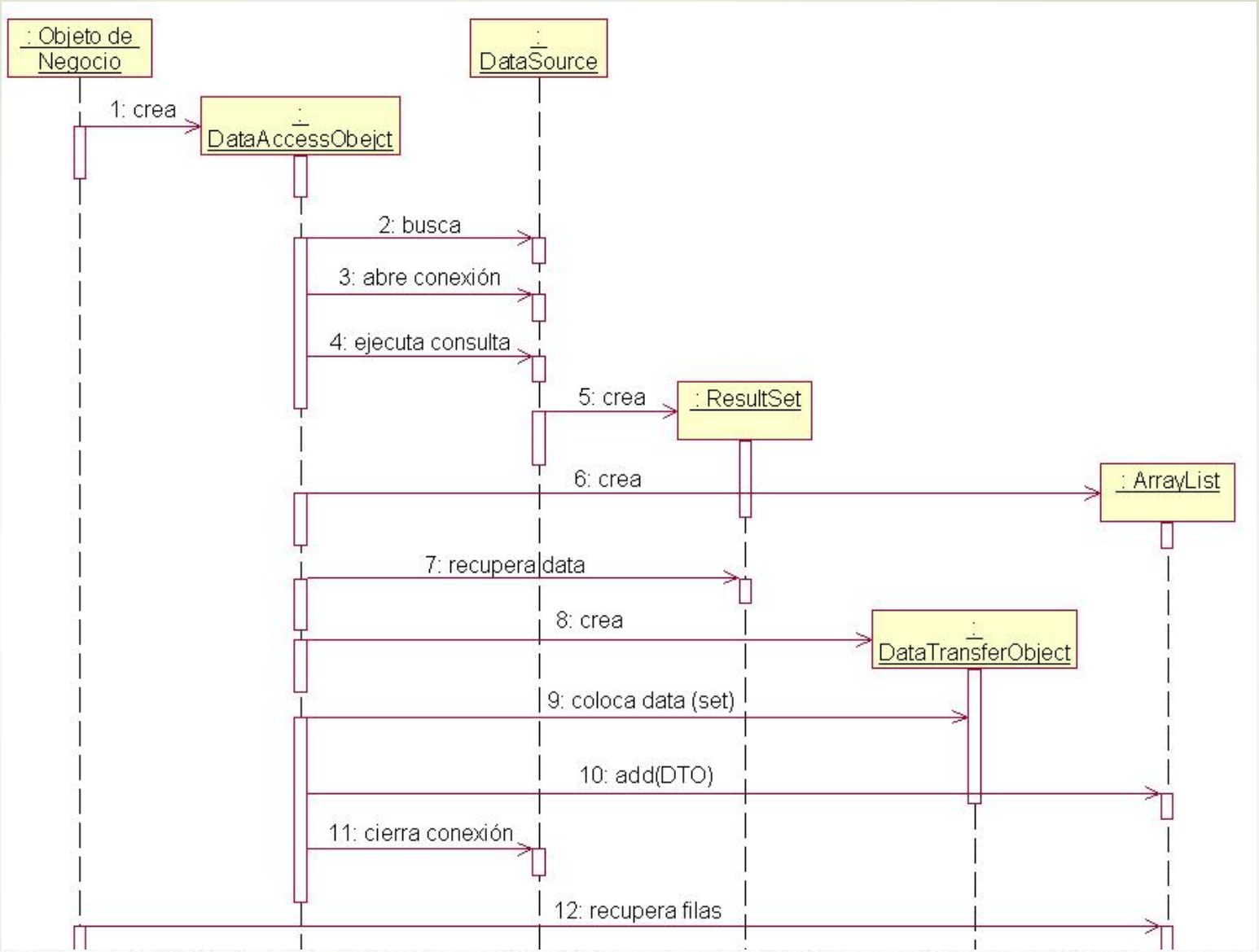
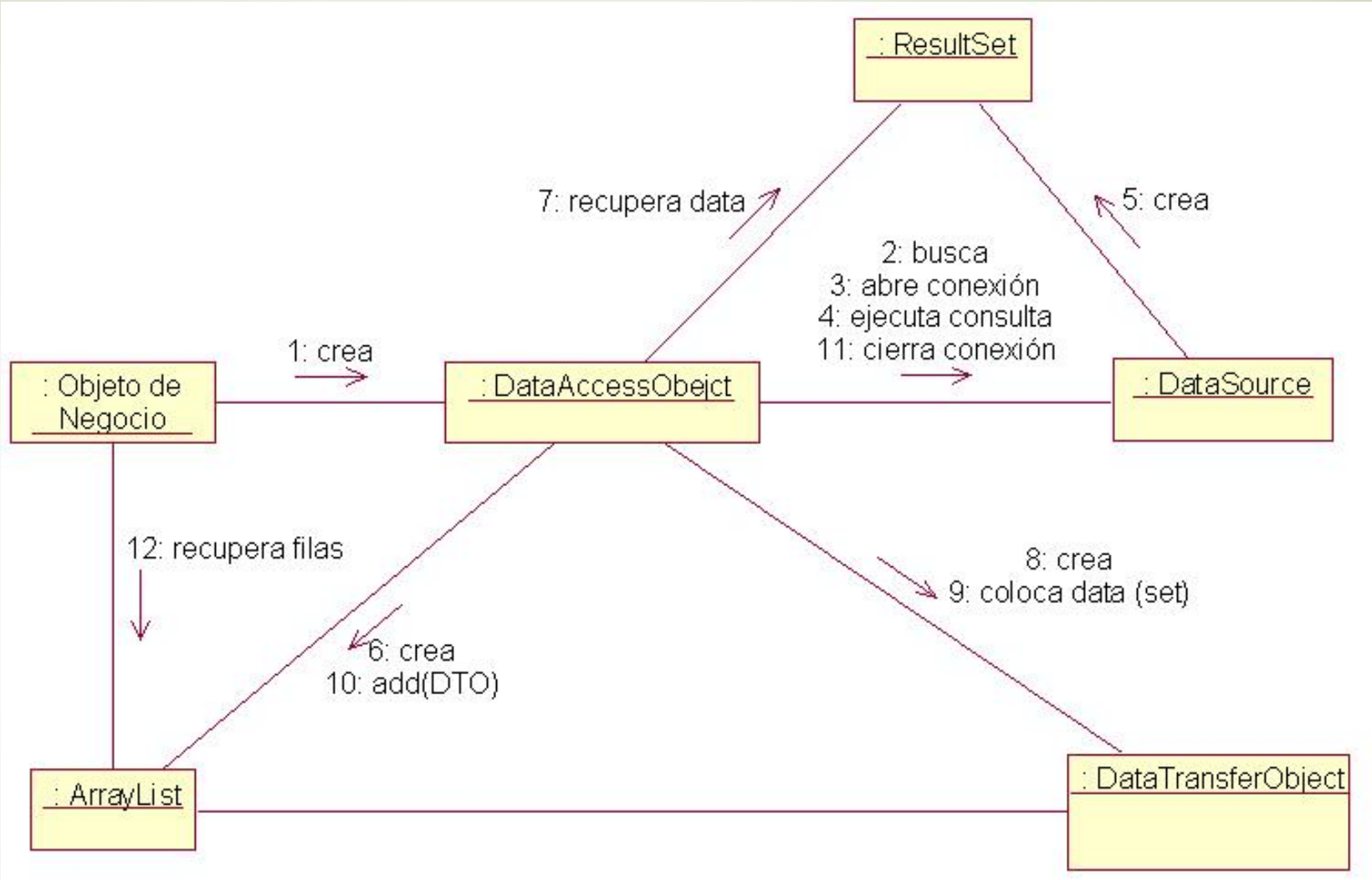


Diagrama de Colaboración



Agenda

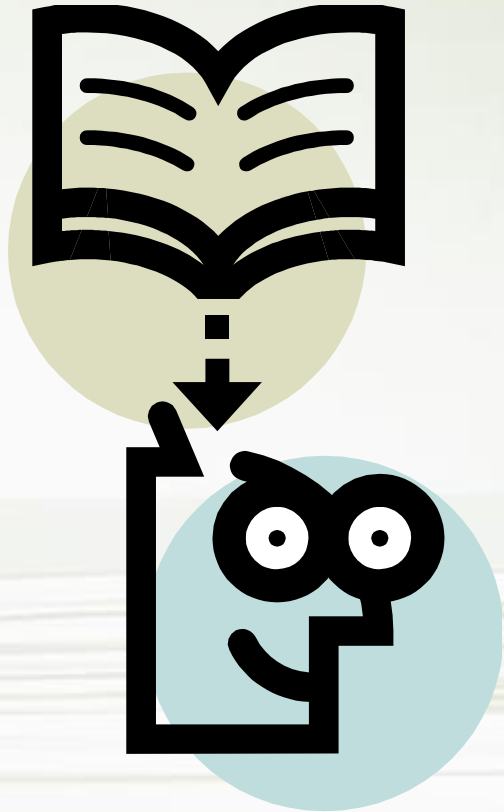
- Motivación para los patrones
- Definiciones de patrones
- Tipos de patrones (Catálogos)
  - De diseño (GoF)
  - De arquitectura (POSA)
  - De plataforma (J2EE)
- Ejemplo (Patrón DAO)
  - Contexto y definiciones
  - Ejemplos
- Conclusiones

Breves Notas

- ¿Patrones como moda? Mejor como Intención.
- ¿Patrones como solución? Mejor como núcleo de soluciones.
- ¿Patrones como normas? Mejor como sugerencias.

Conclusiones

¿¿¿¿



????