



Patrones de diseño

Sesión 1:

Arquitecturas J2EE y patrones de diseño

- **Filosofía del módulo**
- **Conceptos básicos de arquitecturas J2EE**
 - **Aplicaciones monolíticas**
 - **Modelo de capas**
- **Modelos de arquitectura para J2EE**
- **Patrones de diseño**
 - **Patrones “genéricos”**
 - **Patrones J2EE**

- **Filosofía del módulo**
- **Conceptos básicos de arquitecturas J2EE**
 - **Aplicaciones monolíticas**
 - **Modelo de capas**
- **Modelos de arquitectura para J2EE**
- **Patrones de diseño**
 - **Patrones “genéricos”**
 - **Patrones J2EE**

- En J2EE disponemos de un extenso conjunto de tecnologías

JAXP EJB JDBC
JSP JavaMail
JAAS JTA
Servlets

- ¿Cómo usarlos de manera razonable para construir una aplicación J2EE?
- ¿Cómo estructurar la arquitectura de una aplicación J2EE?

- **Problema: ¿Qué significa “razonable”?**
 - **No hay una solución universal aplicable a todos los casos y en todas las ocasiones**
 - **Hasta cierto punto, es un asunto opinable...**

- **Fuente de información básica en arquitecturas J2EE**
 - **Sun Java Blueprints (<http://java.sun.com/blueprints>)**
 - **Guías de arquitectura**
 - **Código de ejemplo**
 - **Patrones de diseño**
 - **...**

- **Filosofía del módulo**
- **Conceptos básicos de arquitecturas J2EE**
 - **Aplicaciones monolíticas**
 - **Modelo de capas**
- **Modelos de arquitectura para J2EE**
- **Patrones de diseño**
 - **Patrones “genéricos”**
 - **Patrones J2EE**

Aplicaciones monolíticas

```
<html>
<head>
  <title>TW</title>
</head>
<body>
<h2>CARTELERA DE HOY: <%=df.format(fechaHoy)%> </h2>

<%
  // se crea la consulta sql:
  misql="select * from peliculas pe, pases pa where pe.id=pa.pelicula ";
  misql+=" and date_format(pa.fecha,'%d/%m/%Y')=""+df.format(fechaHoy);
  misql+=" order by pe.id,pa.hora";
try{
// SE ESTABLECE LA CONEXIÓN CON LA BD:
  Class.forName("com.mysql.jdbc.Driver");
  connection = DriverManager.getConnection(jdbc:mysql//localhost:3306/cines,"","");
// CONEXIÓN CON LA BD ESTABLECIDA. SE EJECUTA LA CONSULTA:
  stmt = connection.createStatement();
  rs=stmt.executeQuery(misql);
%>
<table border=1>
<tr><td colspan="2">Película</td><td>Pase</td></tr>
<%
  // ahora se va a recorrer el recordset. Para cada película se ponen los pases de cada una.
  if (rs.next()) {
```

Cliente

Presentación (visualización de datos,
navegación,...)

Negocio (lógica)

Integración (acceso a datos)

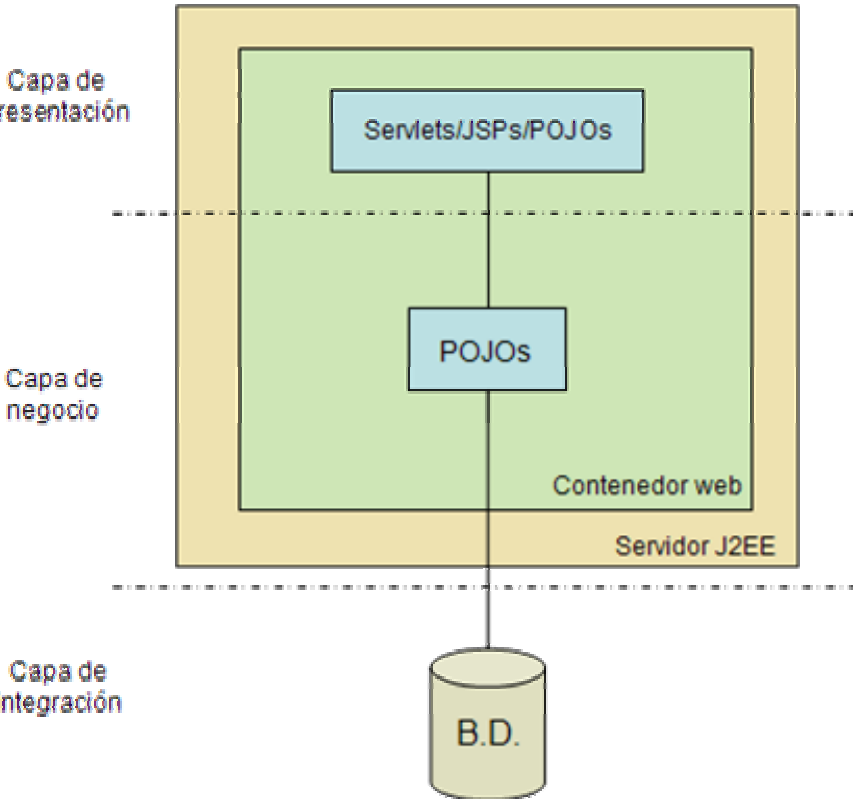
Recursos: (B.D., sistemas no-java,...)

- **Las responsabilidades se dividen en capas**
 - **Arquitectura modular, más fácil de mantener y probar**
 - **Los componentes pueden ser distribuidos más fácilmente**
- **Modelo aceptado por el 100% de la comunidad J2EE (bueno, ¿99.9%?)**

- **Filosofía del módulo**
- **Conceptos básicos de arquitecturas J2EE**
 - **Aplicaciones monolíticas**
 - **Modelo de capas**
- **Modelos de arquitectura para J2EE**
- **Patrones de diseño**
 - **Patrones “genéricos”**
 - **Patrones J2EE**

- **En base a:**
 - **Separación física de las capas: locales vs. distribuidas**
 - **Clientes de la aplicación: navegadores, *rich clients*, servicios web,...**
 - **Gestión de transacciones, concurrencia y seguridad: EJBs vs. NO EJBs**

Arquitectura local sin EJBs

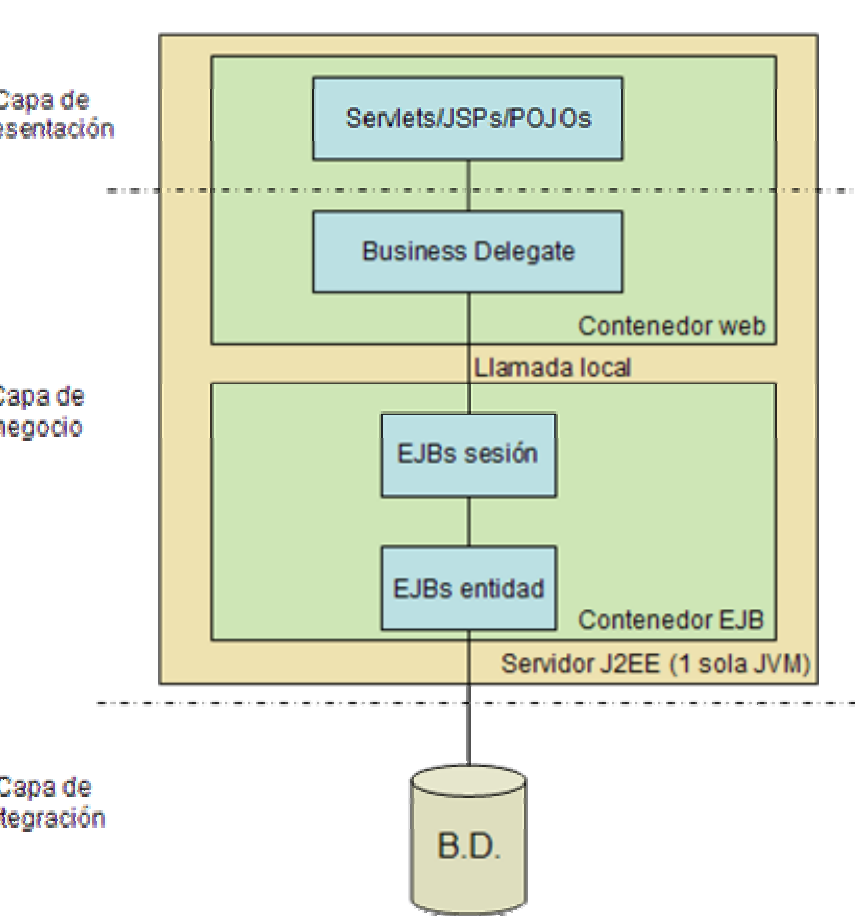


- **Velocidad**
- **Pocos requerimientos de implementación y ejecución**
- **Sencilla de testear**



- **Solo clientes web**
- **¿Escalabilidad?**
- **Gestión manual de transacciones, seguridad, concurrencia y persistencia**

Arquitectura local con EJBs

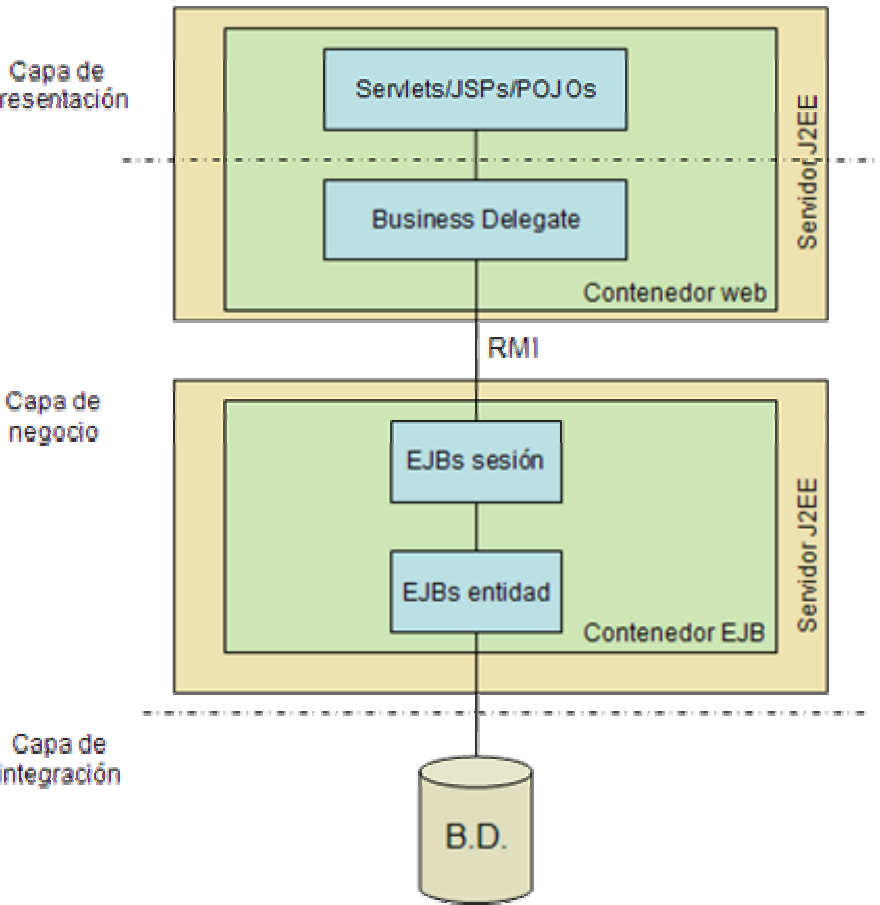


- **Velocidad (llamadas locales)**
- **Servicios del contenedor EJB: transacciones, seguridad, concurrencia y persistencia**



- **Solo clientes web**
- **¿Escalabilidad?**
- **Despliegue y prueba más complejos**

Arquitectura distribuida con EJBs (“de libro”)

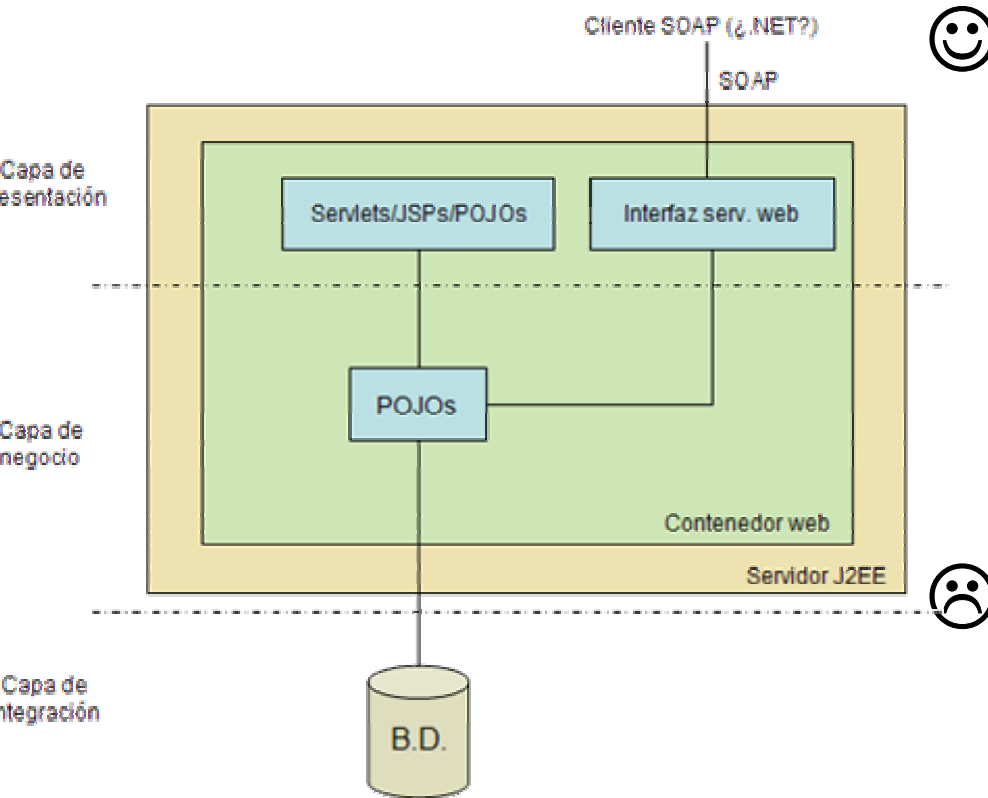


- Cualquier cliente
- Servicios del contenedor EJB: transacciones, seguridad, concurrencia y persistencia
- Distribución de componentes



- Compleja en desarrollo, despliegue y prueba
- Comunicaciones costosas (RMI)

Interfaz añadido para servicios web (+ modelo 1,2 o 3)



- **Cualquier cliente (incluso .NET!!!)**
- **Cientes no navegadores SIN necesidad de EJBs**
- **Uso de SOAP: menos eficiente y más complejo que RMI**

- **Filosofía del módulo**
- **Conceptos básicos de arquitecturas J2EE**
 - **Aplicaciones monolíticas**
 - **Modelo de capas**
- **Modelos de arquitectura para J2EE**
- **Patrones de diseño**
 - **Patrones “genéricos”**
 - **Patrones J2EE**

- **En el desarrollo de aplicaciones J2EE (¡y no J2EE!) se presentan una y otra vez los mismos problemas**
 - Autenticación del cliente
 - Persistencia de datos
 - Separación entre presentación y control,...
 - ...

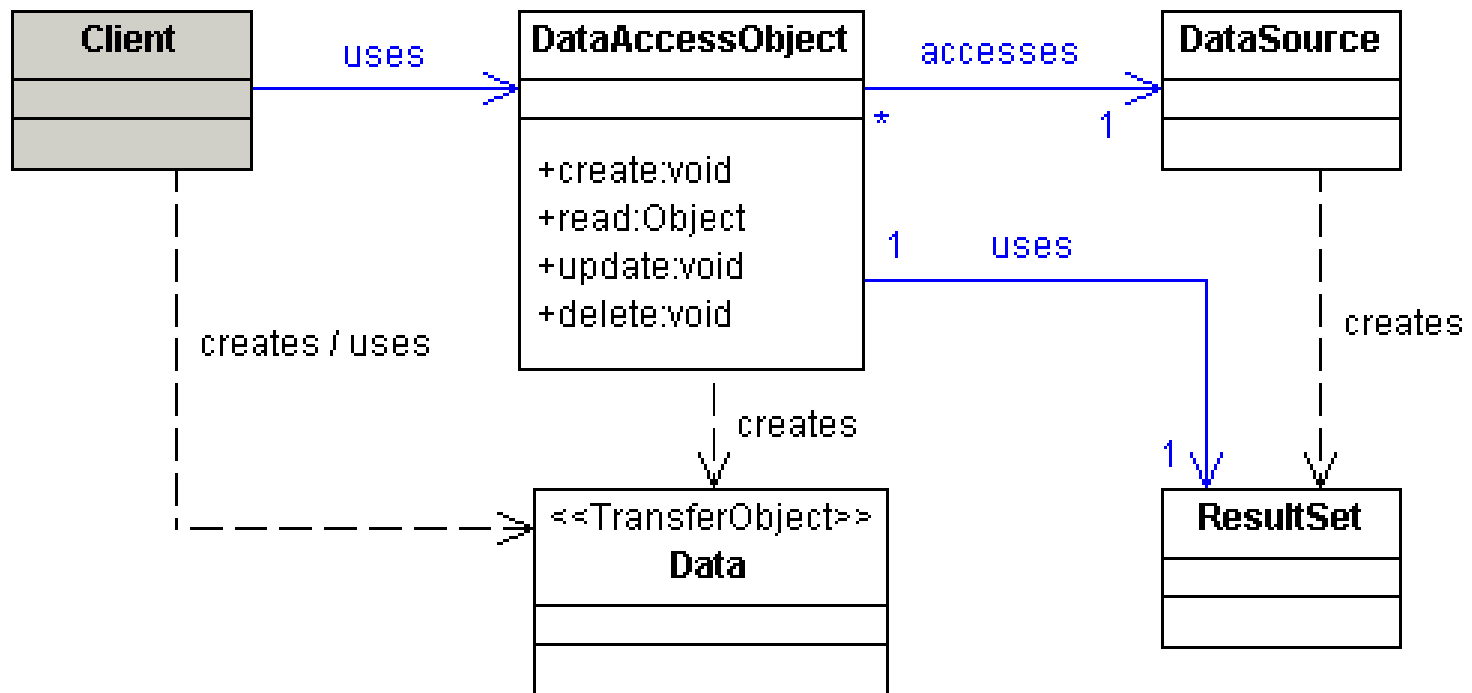
- **En lugar de solucionarlos siempre partiendo de cero, pueden utilizarse patrones de diseño**

- **Un patrón (*pattern*) es una solución ya probada y aplicable a un problema que se presenta una y otra vez**

Ejemplo de patrón: Data Access Object



- Encapsula el acceso a una fuente de datos (normalmente una B.D.)

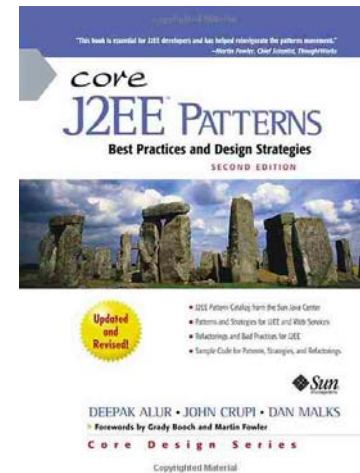
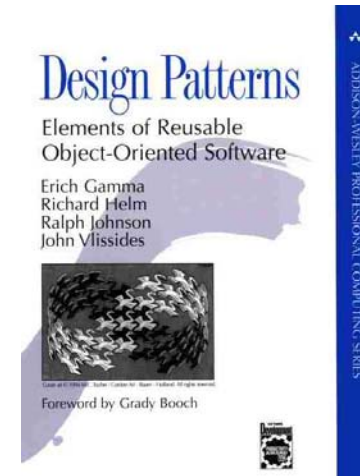


¿Por qué patrones?



- **Están probados**
 - No hace falta “reinventar la rueda”
- **Son reutilizables**
 - Se pueden usar en muchas aplicaciones
- **Son expresivos**
 - Ayudan a crear un vocabulario común para el equipo de desarrollo: (“¿Qué crees, deberíamos hacer esto con un DAO o mejor CMP?”)

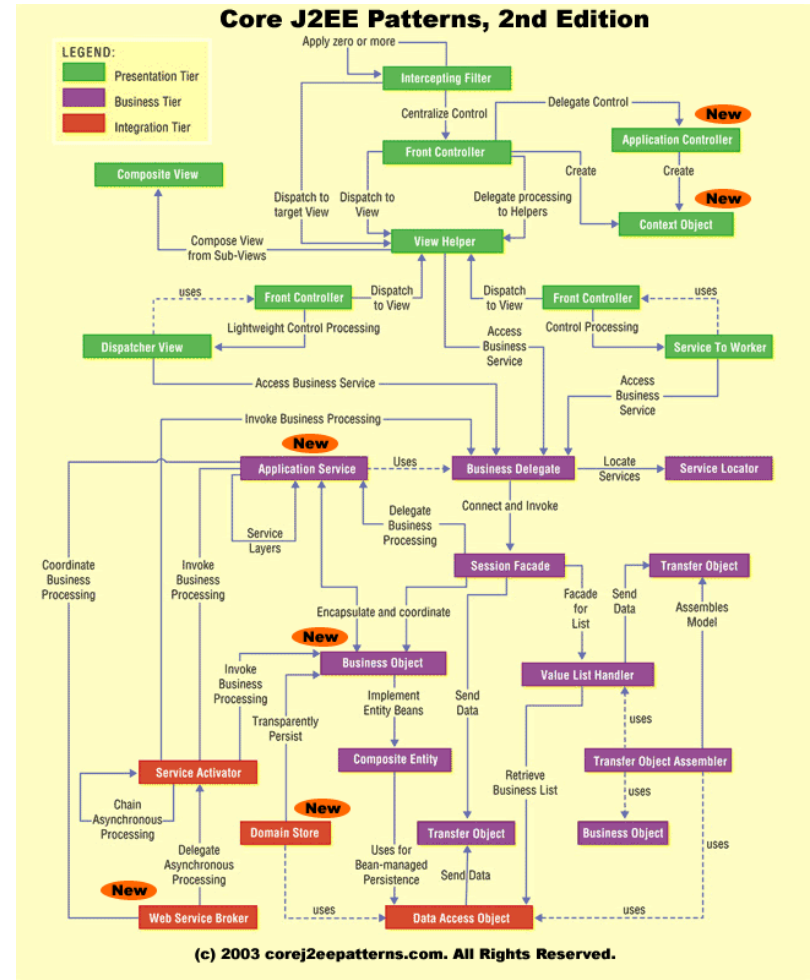
- Los patrones fueron popularizados en el libro “Design Patterns” por Gamma, Helm, Johnson y Vlissides (a partir de aquí, el “Gang of Four” [GoF])
 - Patrones genéricos aplicables a cualquier lenguaje OO
- En J2EE, el movimiento comenzó con “Core Design Patterns”
 - Patrones específicos para aplicaciones J2EE



Esquema de patrones J2EE



- 21 patrones, divididos según las capas
- A todo esto habría que añadir los 23 patrones del GoF más los que algún aburrido inventa cada día...



Peligros de los patrones: overengineering



LOS GRANDES INVENTOS de TBO



Y ¿SABEN LOS CONDUCTORES QUÁN MOLESTO ES CONDUCIR CON EL SOL DE FRENTE. LO MALO ES QUE, DEBIDO A LAS CURVAS, EL DESLUMBRAMIENTO APARECE DE IMPROVISO Y ES NECESARIO REMEDIARLO INMEDIATAMENTE.



ENTONCES EL QUE CONDUCE EMPIEZA A BUSCAR FANOSAMENTE LAS GAFAS DE SOL EN LA GUANTERA, POR LOS BOLSILLOS... Y GENERALMENTE NO LOS ENCUENTRA NI EN UNO NI EN OTRO SITIO. ¿QUE HACER ENTONCES? EL PELIGRO ES EVIDENTE, TANTO POR EL PROPIO DESLUMBRAMIENTO COMO POR LA DISTRACCIÓN QUE OCASIONA A BÚSQUDA.

PARA ESTOS CASOS NUESTRO SAO PROFESOR ACONSEJA QUE SE LEVE MONTADO EN TODOS LOS COE EL SISTEMA DE CORTINA ANTI-DESLUMBRANTE, POLARIZADA Y ANTI-REFLECTANTE DE PUESTA A PUNTO INMEDIATA, CON LA CUAL ES COMO SI LA TOTALIDAD DEL PARABRISAS SE TRANSFORMASE EN UNAS MONUMENTALES GAFAS DE SOL.

CUANDO EL SOL APAREZCA INOPORTUNAMENTE, SE ACCIONARA LA PALANQUITA (C), CON ELLO SE LEVANTA LA TAPA TRASERA (D) DEJANDO VISIBLE EL PLÉTANO QUE EL MONO HAMBRIENTO NO DUDARÁ EN AGARRAR PARA COMERSE, COMO EL PLÉTANO ESTARÁ SUJETO AL EXTREMO DEL CABLE (E) EL TIRON SERÁ TRANSMITIDO POR LAS POLEAS (F) HASTA EL TECHO DEL COCHE EN DONDE HACE DESCENDER (A) QUIZA DE CORTINA) POR ENCIMA DEL PARABRISAS, EL POLITENO TRANSPARENTE (G) DEBIDAMENTE PREPARADO PARA CUMPLIR CON LA FUNCIÓN ANTI-DESLUMBRANTE QUE LE ESTI ENCOMENDADA.

EL SISTEMA SE BASA EN LEVAR SIEMPRE UN MONO (A) EN UNA JAULA (B) SITUADA EN LA PARTE TRASERA DEL COCHE. EL MONO HA DE ESTAR NECESARIAMENTE EN AVANSA DESDE LA VISERA DEL VIAJE.



- **Recogido en “Core J2EE Patterns”**
(<http://www.corej2eepatterns.com>)
- **Veremos algunos patrones básicos del catálogo (no todos)**
- **Los organizaremos según la arquitectura:**
 - **Patrones apropiados para arquitecturas sin EJBs**
 - **Patrones apropiados para arquitecturas distribuidas y/o con EJBs**
 - **Patrones apropiados para servicios web**
- **Dentro de cada arquitectura, iremos por capas**

- **Cada patrón del catálogo se organiza según la siguiente estructura:**
 - **Problema a resolver**
 - **Aplicabilidad: condiciones en las que se aconseja el patrón**
 - **Solución: descripción del patrón**
 - **Estrategias de implementación**
 - **Consecuencias positivas y negativas**