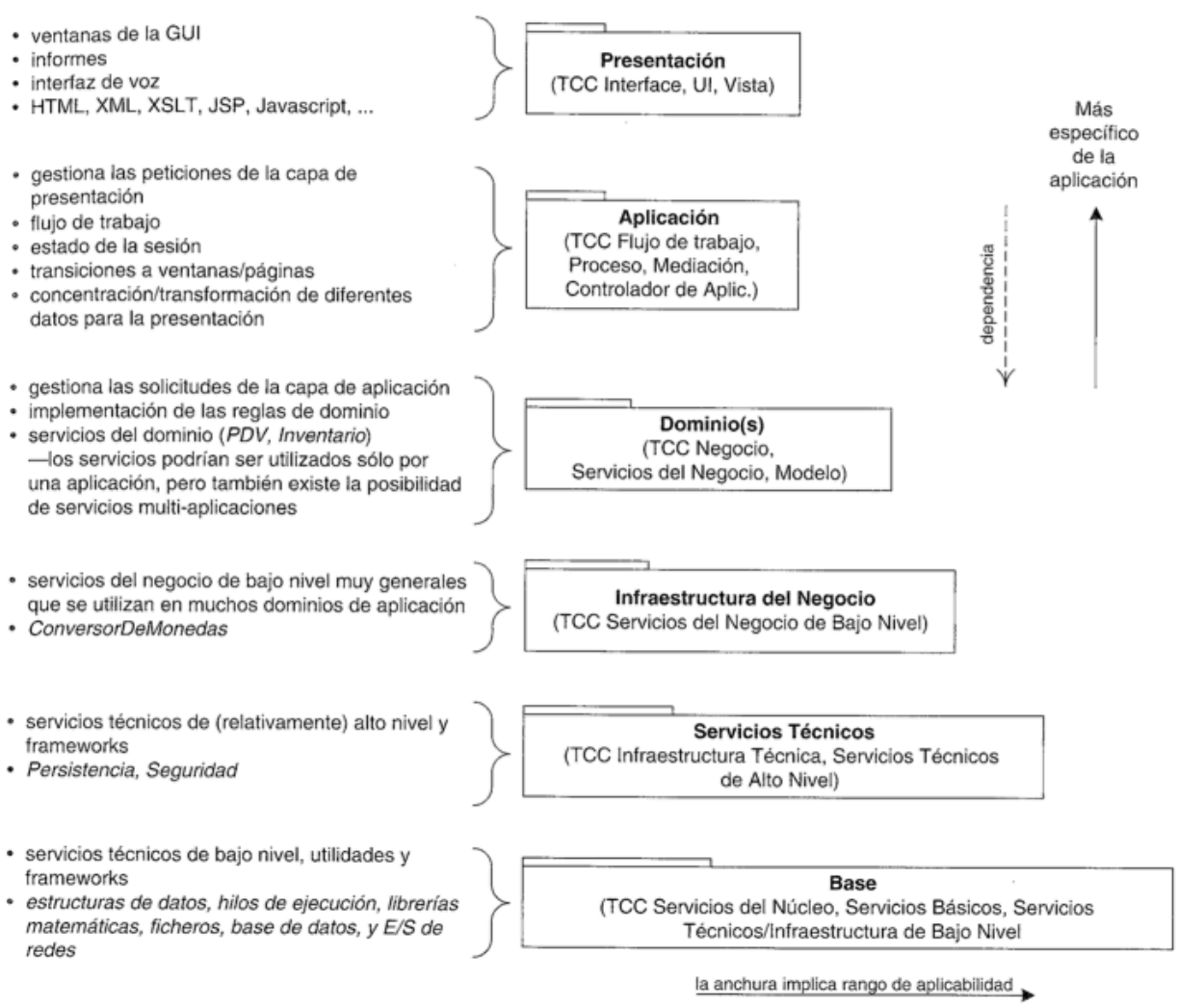
PATRÓN DE ARQUITECTURA: CAPAS (Layers)

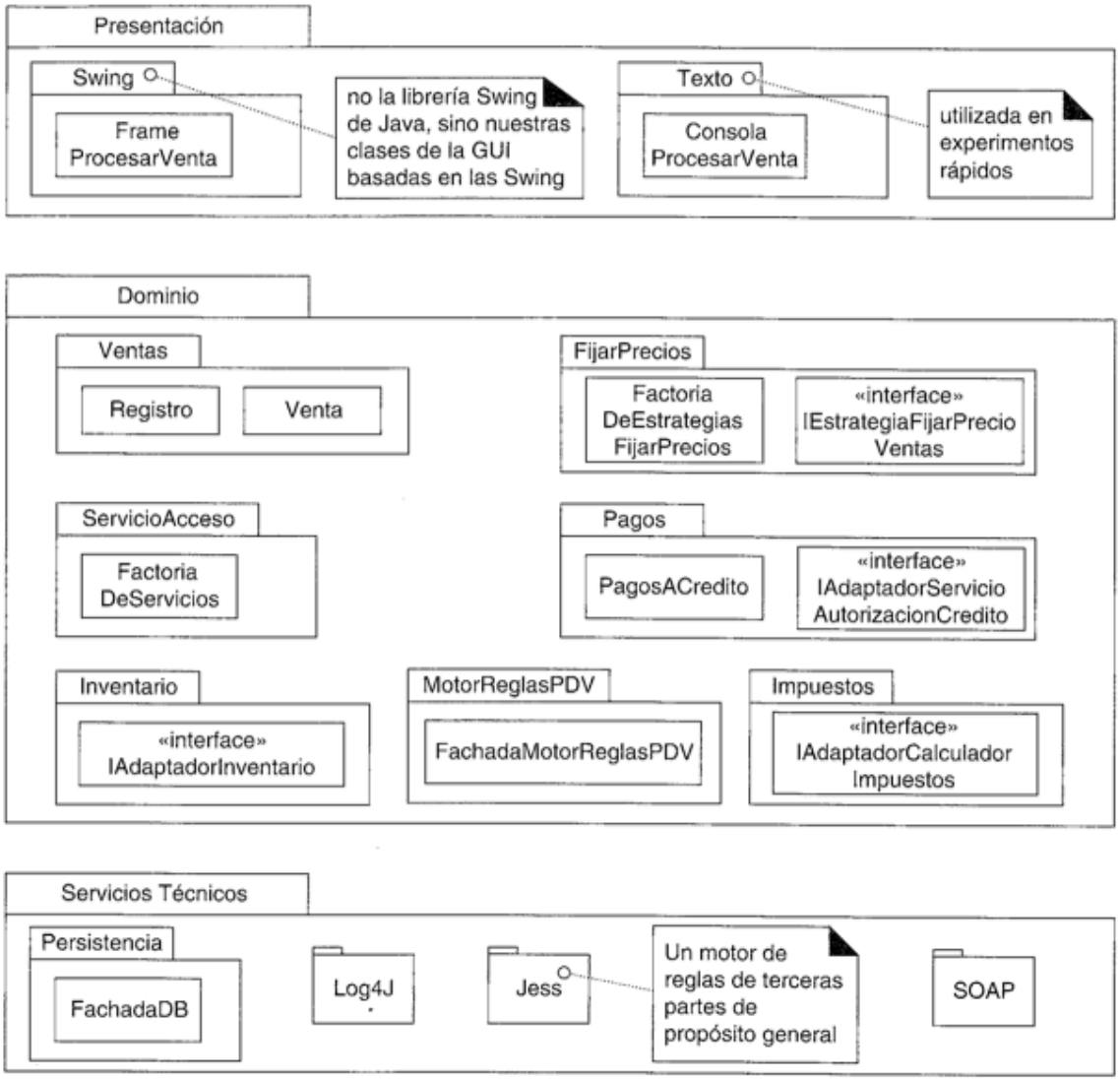
SOLUCIÓN:

- Organizar la estructura lógica de un sistema en **capas separadas con responsabilidades distintas y relacionadas**, con una separación clara y **cohesiva** de intereses como que las capas bajas son servicios generales de bajo nivel, y las capas más altas son más específicas de la aplicación.
- La **colaboración y el acoplamiento es desde las capas más altas hacia las más bajas:** se evita el acoplamiento de las capas más bajas a las más altas.

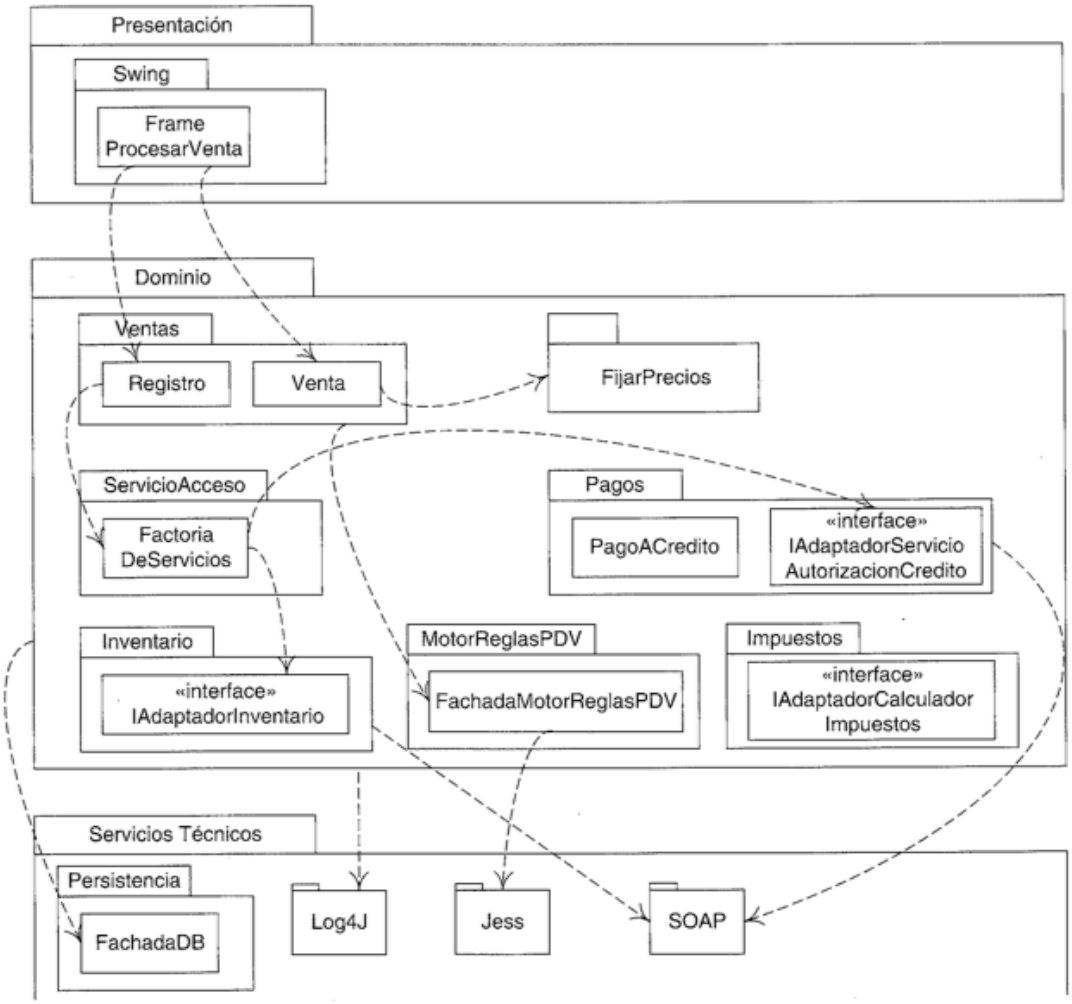
CAPAS COMUNES EN UNA ARQUITECTURA LÓGICA DE UN SISTEMA DE INFORMACIÓN:



VISTA LÓGICA PARCIAL DE LAS CAPAS EN LA APLICACIÓN NuevaEra

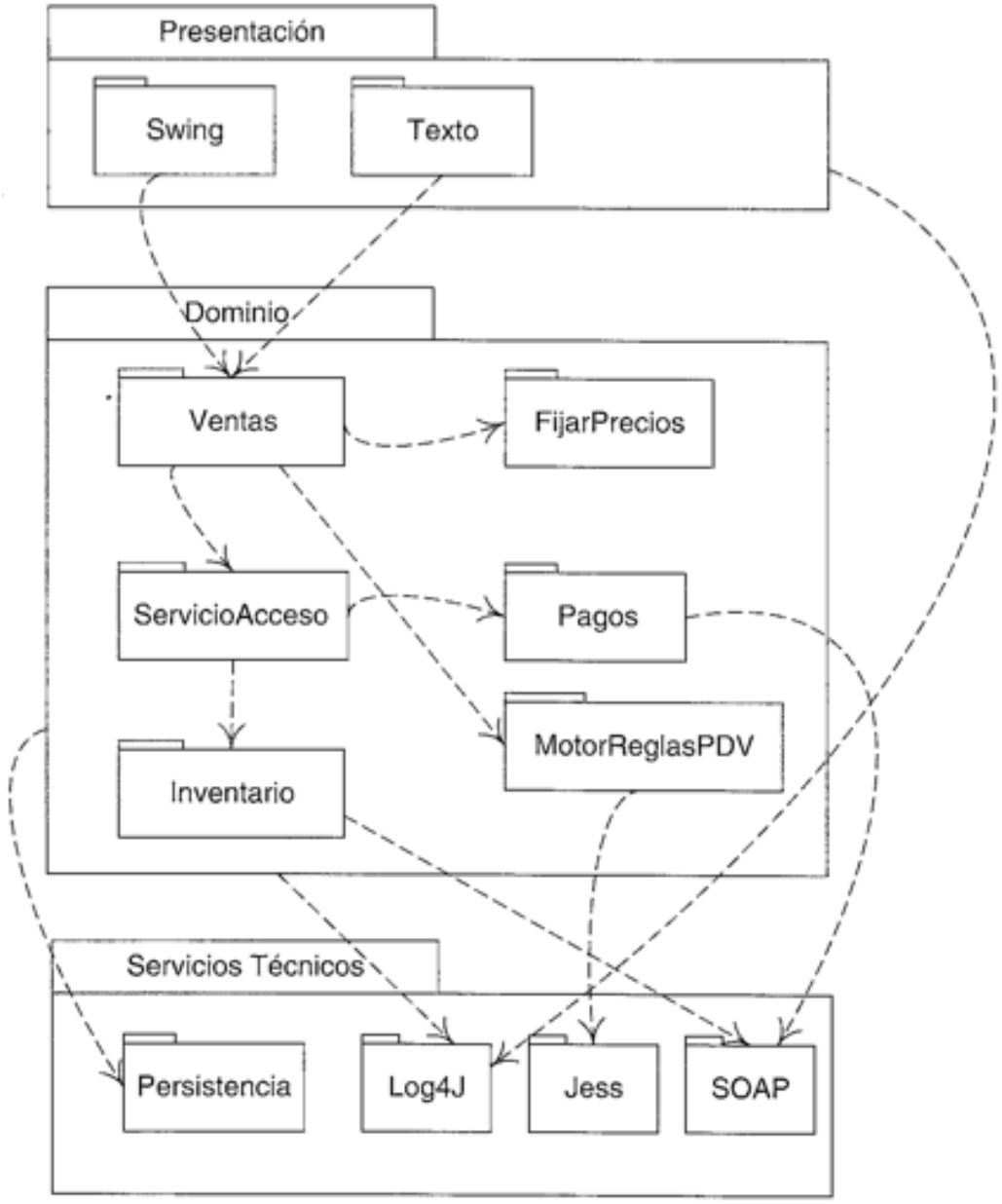


ACOPLAMIENTO PARCIAL ENTRE CAPAS Y PAQUETES



Notación UML:

- Observar que **se pueden utilizar líneas de dependencia para mostrar el acoplamiento entre los paquetes** o los tipos de paquetes. Se utiliza cuando no interesa especificar las dependencias exactas (visibilidad de atributo, subclase, ...) sino que simplemente interesa resaltar las dependencias.
- **Se usa una línea de dependencia que sale de un paquete** en lugar de desde un tipo específico, como desde el paquete Ventas a la clase FachadaMotorReglasPDV, y del paquete del Dominio al paquete Log4J. Esto es útil cuando o bien no es interesante el tipo concreto del que depende, o bien el comunicador **quiere dar a entender que podrían compartir la dependencia muchos elementos del paquete**.
- Otro uso común del diagrama de paquetes es **ocultar los tipos específicos y centrarse en ilustrar el acoplamiento paquete-paquete**, como en el diagrama parcial de la siguiente figura, que ilustra el estilo probablemente más común del diagrama de la arquitectura lógica en UML: un diagrama de paquetes que normalmente muestra de 5 a 20 paquetes importantes y sus dependencias.

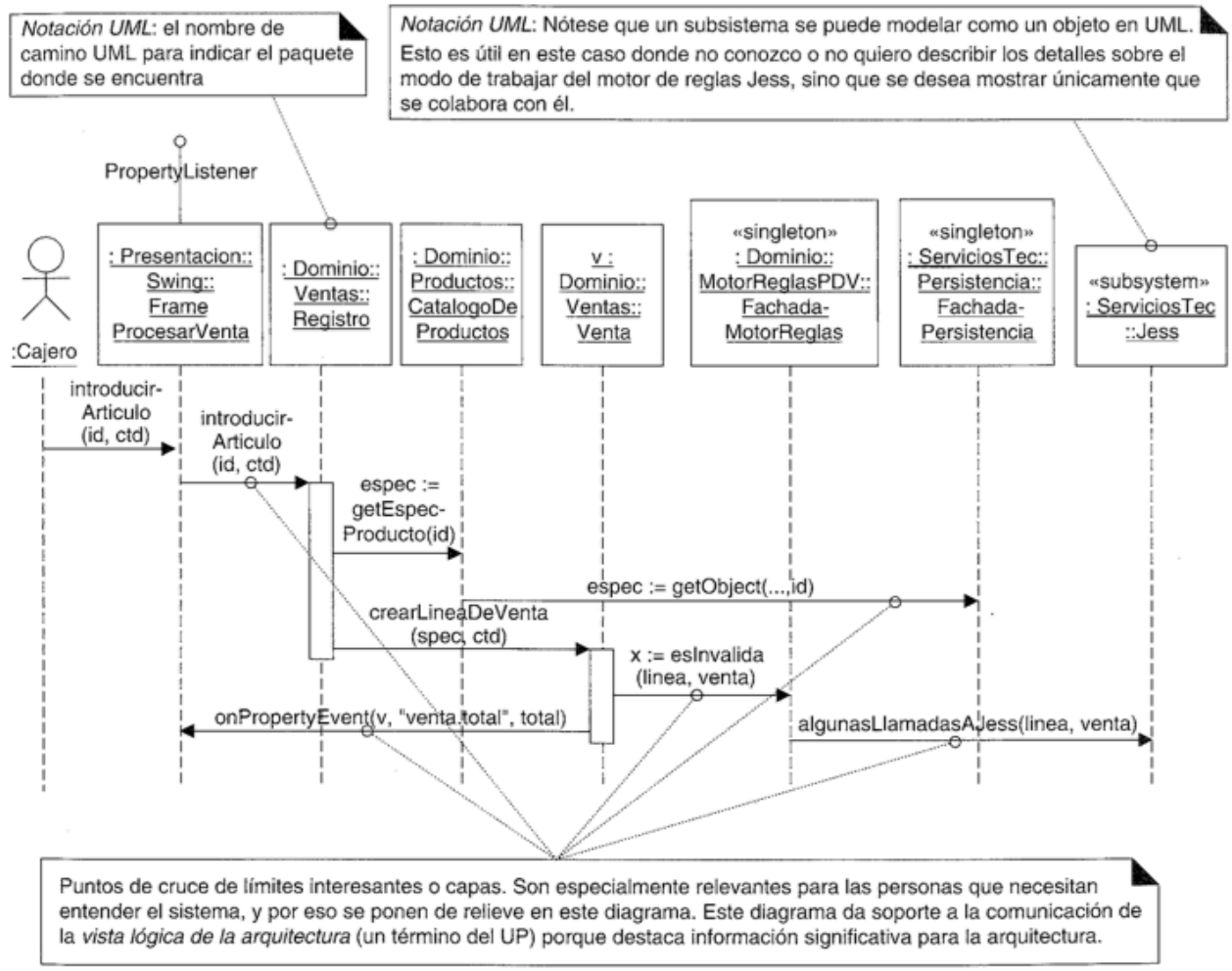


ESCENARIOS DE INTERACCIÓN ENTRE CAPAS Y ENTRE PAQUETES

Los diagramas de paquetes muestran información estática.

Un diagrama de interacción proporciona la información para entender la dinámica del modo en el que se comunican los objetos entre las capas.

Por tanto, es útil contar con un **conjunto de diagramas de interacción** que ilustren los **escenarios más significativos desde el punto de vista de la arquitectura**.



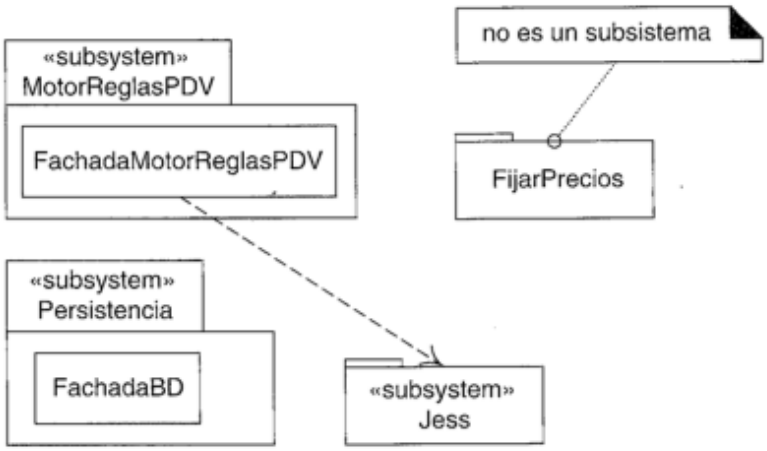
Notación UML:

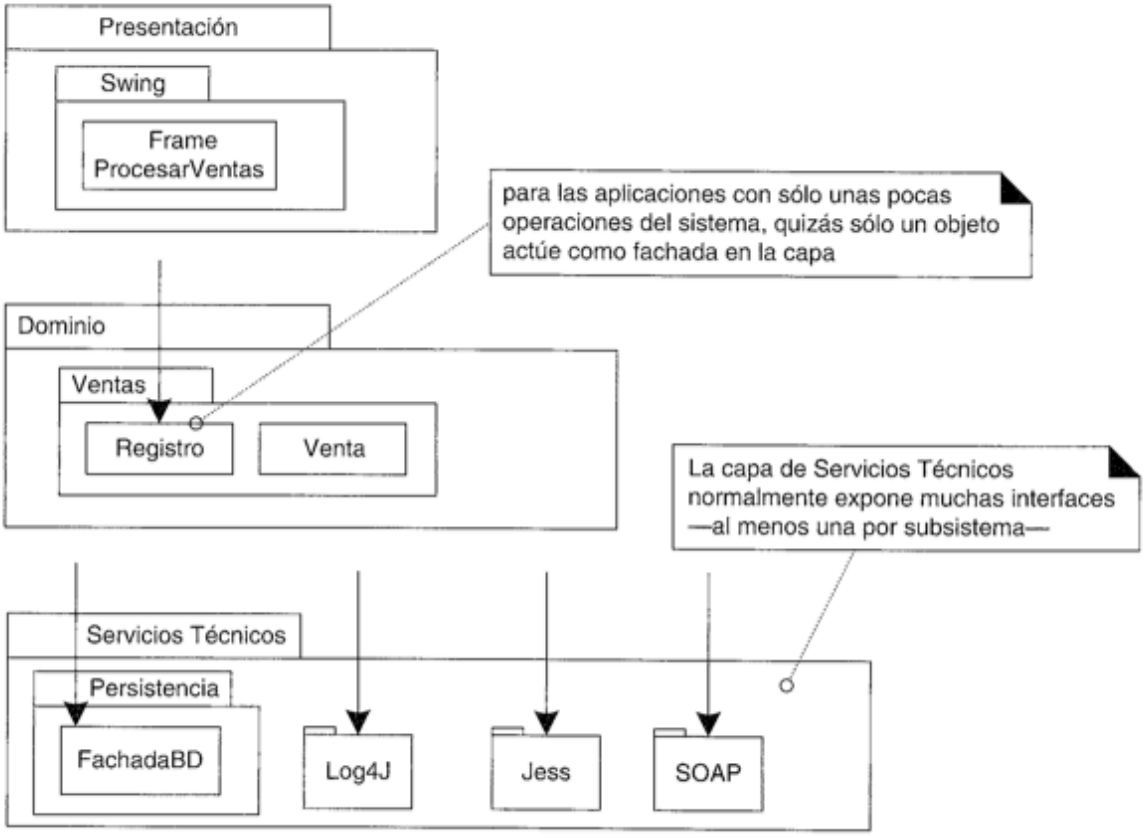
- El paquete al que pertenece el tipo se puede mostrar opcionalmente calificando el tipo con la expresión del **nombre de camino** de UML <NombrePaquete>::<NombreTipo>. Por ejemplo: Dominio::Ventas::Registro.
- Es relevante también el uso del estereotipo **<<subsystem>>.** En UML, un **subsistema** es una entidad discreta que tiene comportamiento e interfaces. Se puede modelar un subsistema como un tipo especial de paquete, o como un objeto (como se muestra en el diagrama de interacción), que es útil cuando uno quiere mostrar las **colaboraciones entre subsistemas**.
- El diagrama no muestra algunos mensajes, como ciertas colaboraciones de la Venta, para poner de relieve las **interacciones significativas para la arquitectura.**

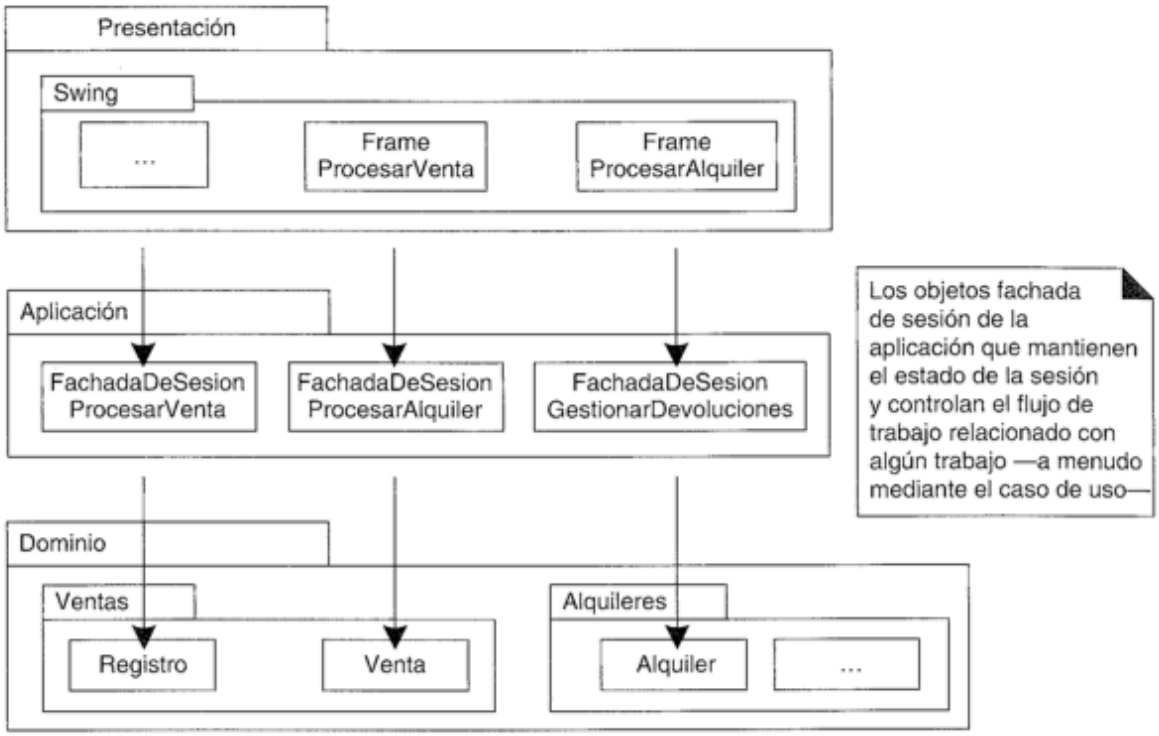
COLABORACIONES

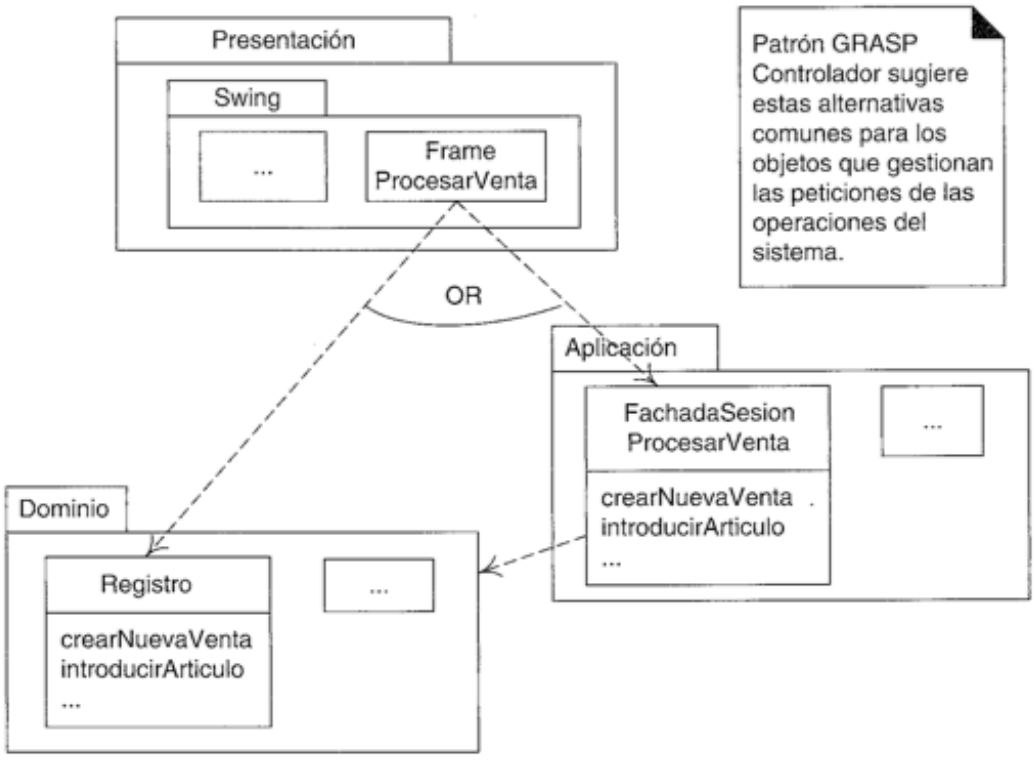
Dos decisiones de diseño al nivel de arquitectura son:

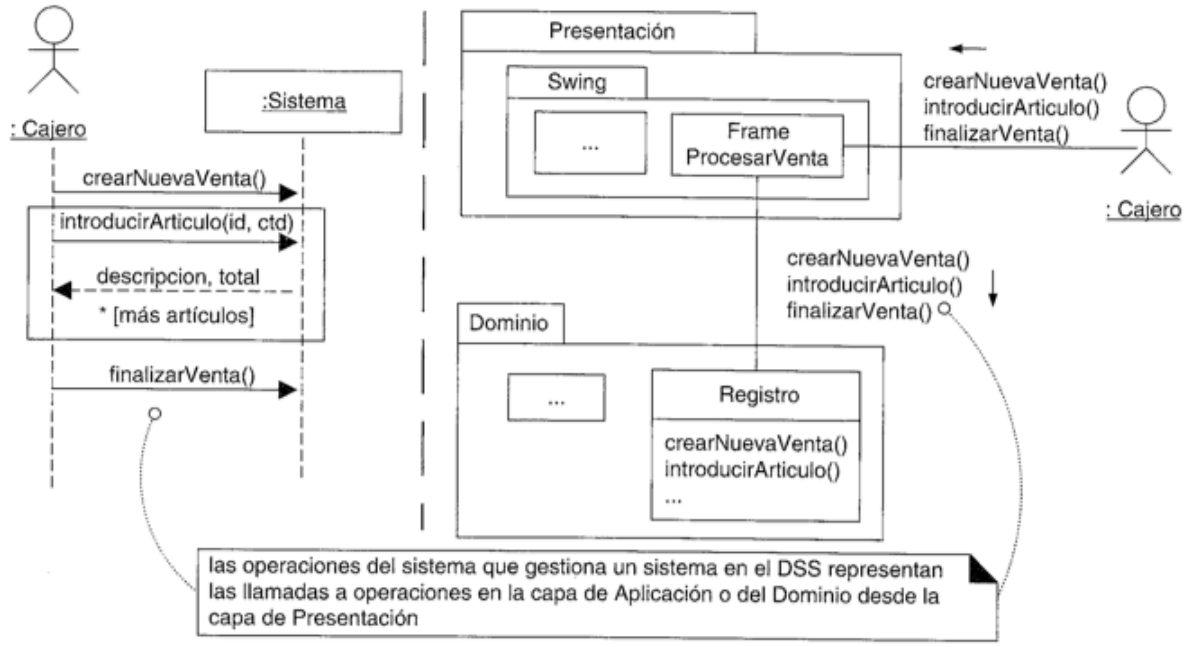
1. ¿Cuáles son las partes importantes?
2. ¿Cómo se conectan?


«subsystem»
MotorReglasPDV
FachadaMotorReglasPDV
no es un subsistema
FijarPrecios
«subsystem»
Persistencia
FachadaBD
«subsystem»
Jess


Presentación
Swing
Frame
ProcesarVentas
para las aplicaciones con sólo unas pocas operaciones del sistema, quizás sólo un objeto actúe como fachada en la capa
Dominio
Ventas
Registro
Venta
La capa de Servicios Técnicos normalmente expone muchas interfaces —al menos una por subsistema—
Servicios Técnicos
Persistencia
FachadaBD
Log4J
Jess
SOAP

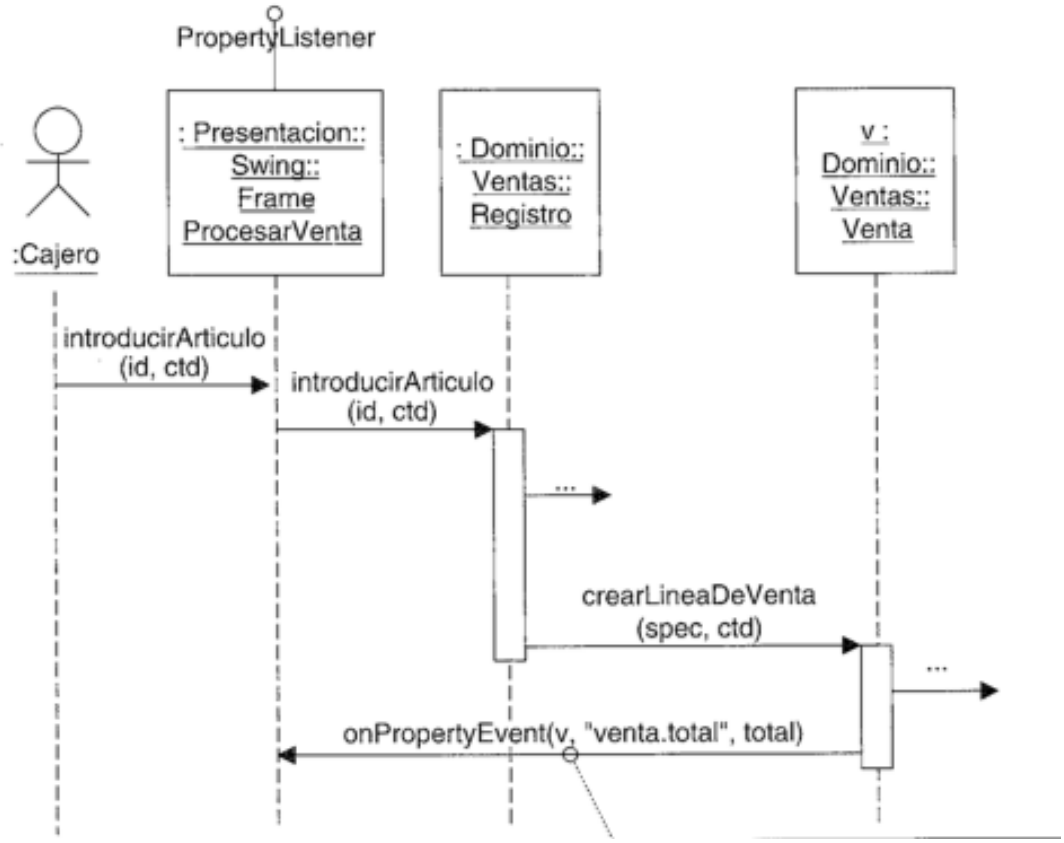

Presentación
Swing
...
Frame ProcesarVenta
Frame ProcesarAlquiler
Aplicación
FachadaDeSesion ProcesarVenta
FachadaDeSesion ProcesarAlquiler
FachadaDeSesion GestionarDevoluciones
Los objetos fachada de sesión de la aplicación que mantienen el estado de la sesión y controlan el flujo de trabajo relacionado con algún trabajo —a menudo mediante el caso de uso—
Dominio
Ventas
Registro
Venta
Alquileres
Alquiler
...


Presentación
Swing
...
Frame ProcesarVenta
Patrón GRASP Controlador sugiere estas alternativas comunes para los objetos que gestionan las peticiones de las operaciones del sistema.
OR
Aplicación
FachadaSesion ProcesarVenta
crearNuevaVenta
introducirArticulo
...
...
Dominio
Registro
crearNuevaVenta
introducirArticulo
...
...



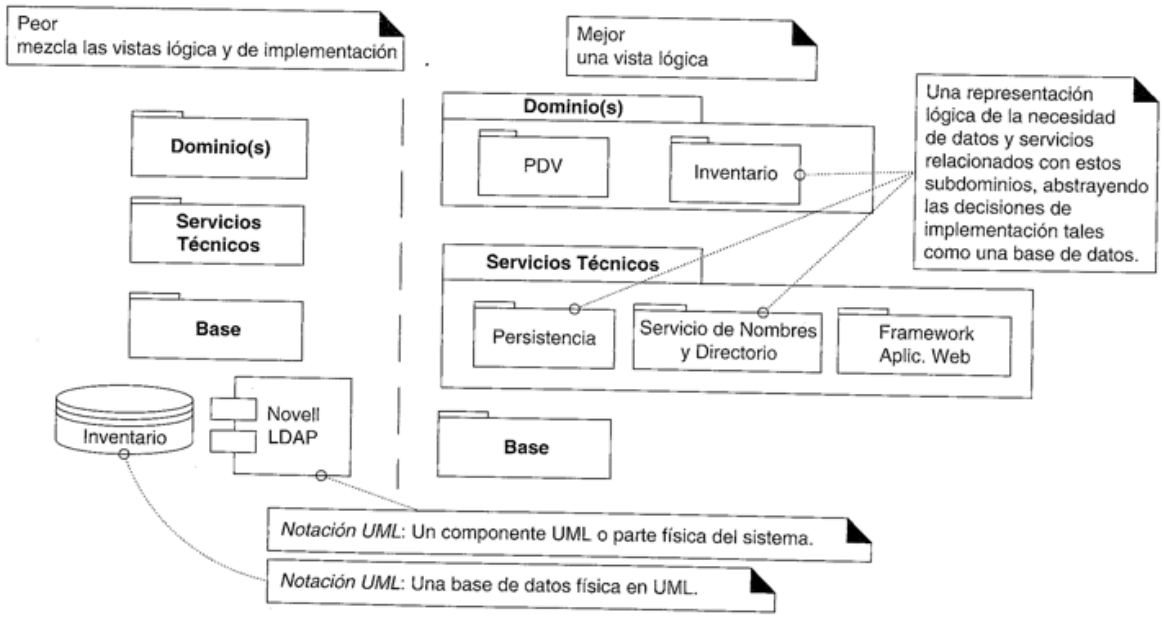
COLABORACIONES ASCENDENTES CON EL PATRÓN OBSERVADOR

Utilización del patrón observador para la comunicación ascendente con la capa de Presentación:



La **colaboración desde las capas más bajas hacia la capa de presentación** normalmente se lleva a cabo mediante el **patrón Observador (Publicar-Suscribir**).

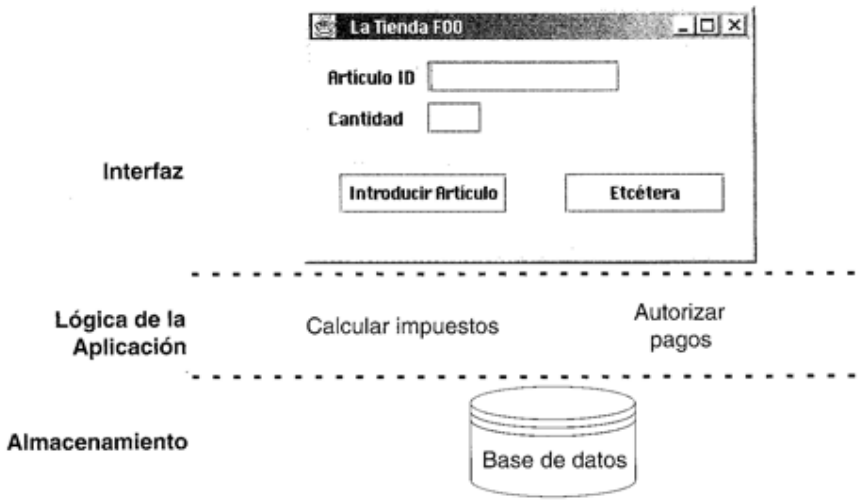
El objeto Venta ha registrado a los subscriptores que son objetos PropertyListener. Uno de ellos es una JFrame Swing de la GUI, pero la Venta no conoce a este objeto como un JFrame de la GUI sino como un PropertyListener.



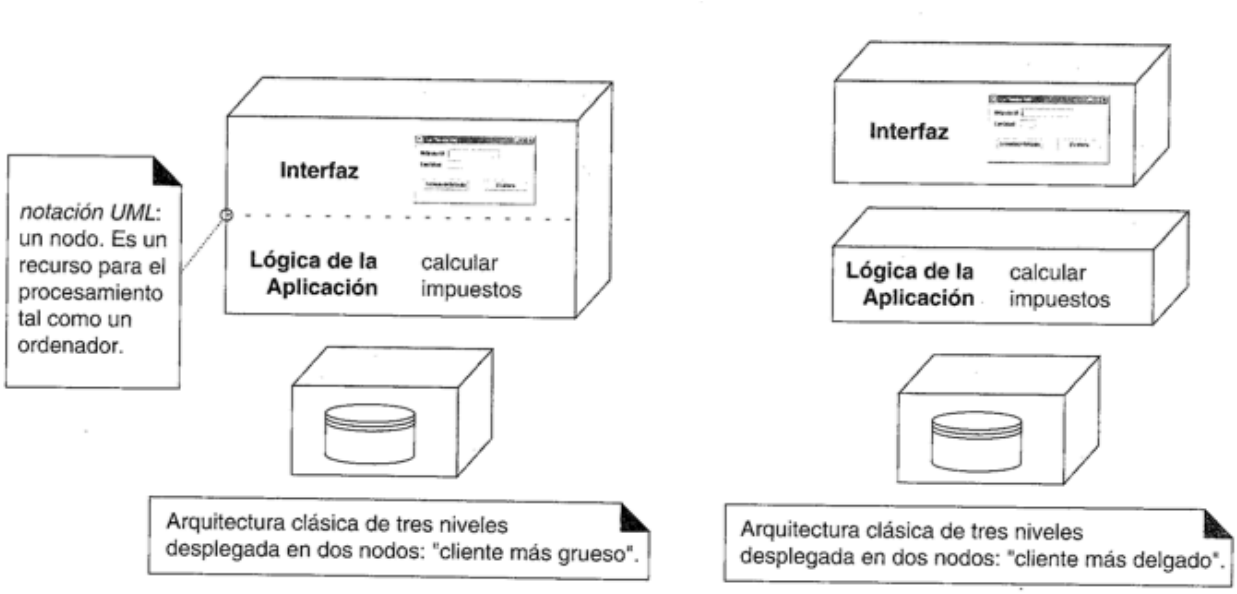
SISTEMAS DE INFORMACIÓN- LA ARQUITECTURA CLÁSICA EN 3 NIVELES

Una descripción clásica de los niveles verticales en la arquitectura de tres niveles es:

1. **Interfaz**: ventanas, informes,…
2. **Lógica de aplicación**: tareas y reglas que dirigen el proceso.
3. **Almacenamient**o: mecanismos de almacenamiento persistente.



El objetivo es la separación de la lógica de la aplicación en un nivel intermedio separado (lo que no quiere decir que tenga que desplegarse en tres nodos).



PRINCIPIO DE SEPARACIÓN MODELO-VISTA

El principio de Separación Modelo-Vista establece que los objetos del modelo (dominio) no deberían conocer directamente a los objetos de la vista (presentación), al menos como objetos de la vista.

Por ejemplo, un objeto Registro o **Venta no debería enviar un mensaje directamente a un objeto ventana de la *GUI*** *FrameProcesarVenta*, pidiéndole que muestre algo, cambie de color, etc.

Este es un principio clave en el patrón Modelo-Vista-Controlador(MVC), que relaciona los objetos de datos (modelos), los elementos gráficos de la GUI (vista), y el manejo de los eventos del ratón y teclado (Controladores).

SEPARACIÓN MODELO-VISTA Y COMUNICACIÓN ASCENDENTE

¿Cómo pueden obtener las ventanas de información lo que tienen que mostrar?

- Normalmente es suficiente que envíen mensajes a los objetos del dominio, preguntando sobre la información que luego mostrarán en elementos gráficos: modelo de escrutinio (**polling**) o tirar-desde-arriba (**pull from above**) para mostrar las actualizaciones.

- A veces es mejor que los pocos objetos del dominio que cambian se comuniquen con las ventanas para que provoque que se actualice la información que muestran cuando el estado del objeto del dominio cambia. En estas situaciones se requiere un modelo empujar desde abajo (**push from below**) para mostrar las actualizaciones.

- Debido al principio de separación Modelo-Vista se necesita establecer una comunicación indirecta desde los objetos inferiores hacia las ventanas.
  Existen dos soluciones comunes:
  1. El **patrón Observador**, haciendo que los objetos de la GUI simplemente parezcan objetos que implementan una interfaz común PropertyListener.
  2. Un **objeto Fachada de Presentación. E**s decir, añadir una fachada en la capa de presentación que recibe las peticiones desde abajo.