# Arquitectura de sistemas de información

# Guion

- Introducción
- Modularidad
- Arquitectura software
- Principios arquitectónicos
- Arquitectura Cliente/Servidor
- Arquitectura basada en capas
- Arquitectura orientada a servicios

# Introducción

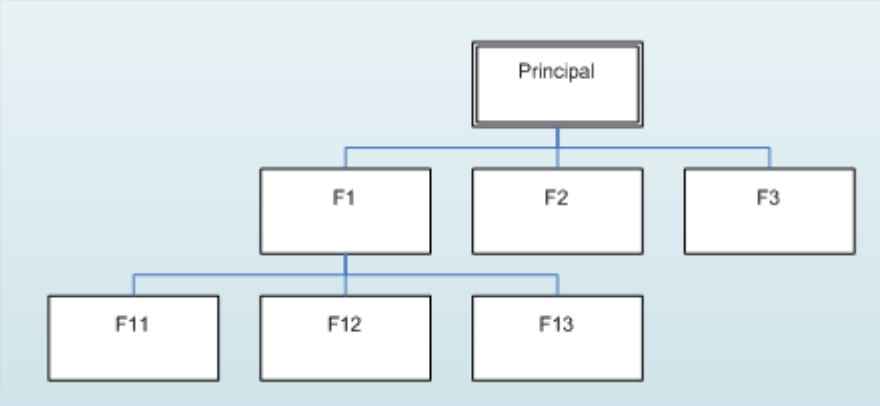
- Los sistemas reales son demasiado grandes y deben ser divididos en partes más pequeñas según el principio de *abstracción.*
  - *Habitualmente usamos la abstracción cuando no es necesario conocer los detalles exactos de cómo funciona algo, porque podemos usarlo en su forma simplificada. [...] (David J. Barnes)*

# Introducción

- Para poder ser capaces de dominar objetos complejos, los humanos ignoramos detalles no esenciales.
- Tratamos con un modelo ideal del objeto, centrándonos en sus aspectos esenciales.
- Por los tanto la *abstracción* consiste en el proceso de excluir detalles no deseados o no significativos al problema que se trata de resolver.
- Se suele aplicar distintos niveles de abstracción.

# Modularidad

- Para construir programas o sistemas recurrimos a la modularidad:
- Se divide el programa o sistema en módulos independientes.



# Modularidad

## Cohesión y acoplamiento

- Cohesión: Un módulo debe tender a hacer sólo una cosa.
- Acoplamiento: Los módulos deben tener la menor dependencia unos de otros. Es decir, los módulos deben ser lo más independientes unos de otros.

# Modularidad

- En consecuencia, la modularidad es la posibilidad de subdividir un programa o sistema en piezas más pequeñas (módulos) cada una de las cuales deben ser tan independiente como sea posible de los restantes módulos.

# Modularidad

## Abstracción procedimental

- Cada módulo debe asegurar que un cambio en su interior no afecta al exterior.
- Si hay un error en un módulo se debe evitar que se propague al exterior.
  - Para alcanzar esto ⇒ Cohesión alta y acoplamiento bajo.

# Modularidad

## Abstracción procedimental

- Una *abstracción procedimental* separa el propósito de un subprograma de su implementación.
- Conocemos de un módulo o subprograma los datos que pide (y cómo), lo que hace y lo que devuelve (y cómo).
- La abstracción procedimental significa centrarse en lo que hace un módulo en lugar de los detalles de cómo se implementan.

# Modularidad

## Estructura de un módulo

- Un módulo se caracteriza:
  - Por su interfaz (parte visible: datos, proced. etc)
  - Por su implementación
- Un módulo es un conjunto de acciones denominadas *funciones* o submódulos que comparten un conjunto de datos comunes llamados *atributos*.
- Los atributos definen el *estado* del módulo.
- Las funciones de un módulo se denominan *primitivas o puntos de entrada*.

# Modularidad

- Ejemplo:
- Módulo: Circunferencia
  - Atributos: centro, radio
  - Funciones:
    - Crear(centro:punto, radio:integer)
    - Area()
    - Cambiar_Radio(radio:integer)
    - Cambiar_Centro(centro:punto)
    - Borrar()

# Modularidad

## Ocultación de información

- La *abstracción de datos* se centra en las operaciones que se ejecutan sobre los datos, en lugar de centrarse en cómo se implementan.
- La abstracción identifica los aspectos esenciales de módulos y estructuras de datos de modo que se puedan tratar como *cajas negras*.
- La abstracción es responsable de identificar la vista externa de los módulos, pero también ayuda a identificar detalles que se deben ocultar (vista privada).
- Lo ideal es que se "vean" *algunas* funciones, y se debe *ocultar datos (estado)*.

# Arquitectura software

- Muchos sistemas reales comparten importantes propiedades subyacentes y dan lugar a problemas de diseño comunes.
- Se pueden describir **modelos** o **paradigmas** de arquitectura software.
- Los modelos están pensados para proporcionar una descripción abstracta, simplificada pero consistente de cada aspecto relevante del diseño de un sistema.

# Arquitectura software

## Servicio

- Un **servicio** es una parte diferente de un sistema de computadores que gestiona una colección de recursos relacionados y presenta su funcionalidad a los usuarios y aplicaciones.
- Es similar a un módulo, pero representa un nivel superior, ya que un servicio suele estar compuesto por varios módulos software y puede que también componentes hardware.
- Así se aplican todos los conceptos de abstracción de datos, específicamente el concepto de *interfaz* pública.

# Arquitectura software

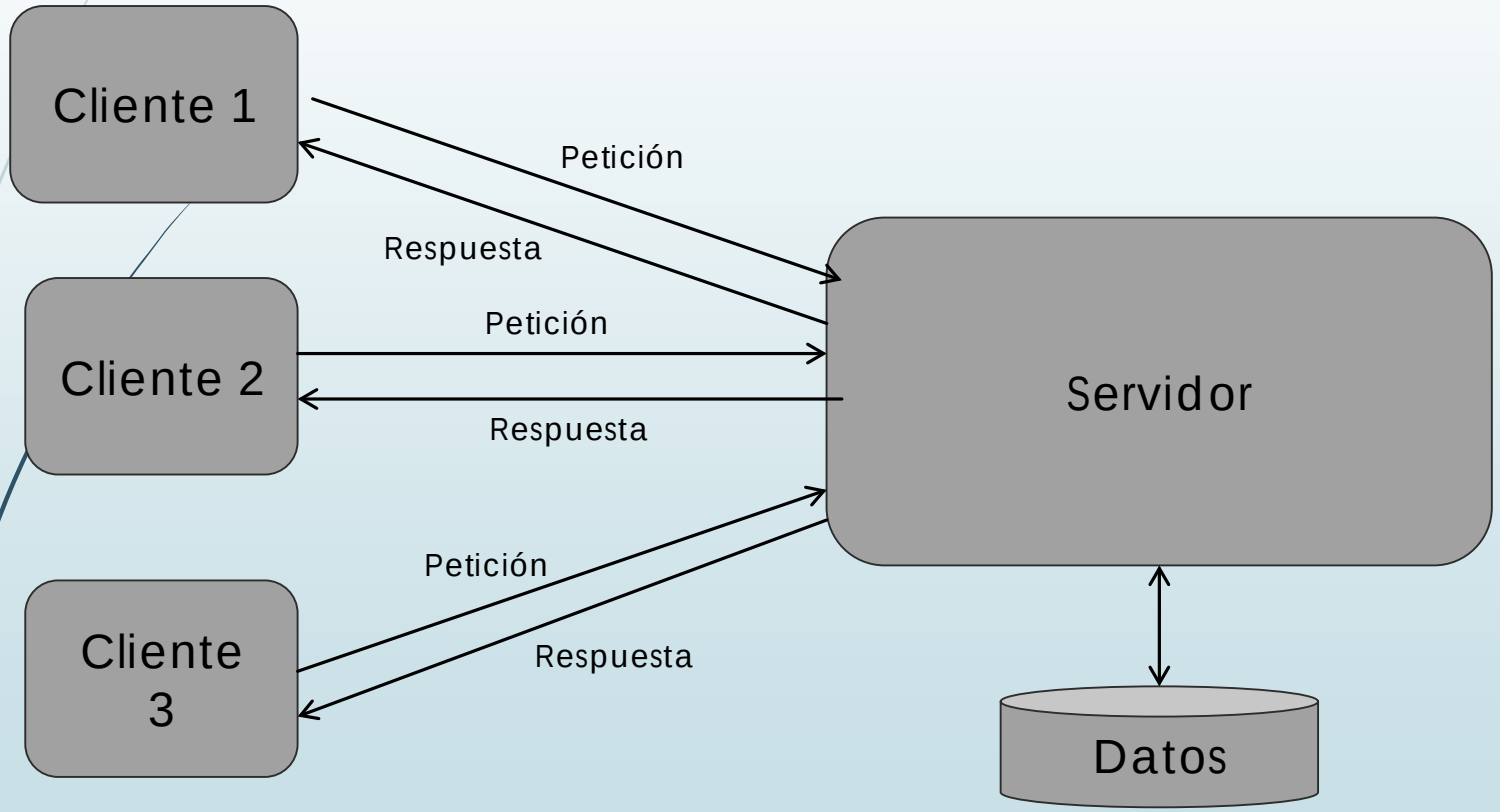
## Servidor / Cliente

- Un **servidor** es un proceso ejecutándose en un ordenador en red que acepta peticiones de procesos ejecutándose en otros computadores para realizar un servicio y responder adecuadamente.
- Los procesos solicitantes se llaman **clientes.**
- Las peticiones se envían por medio de mensajes (similar a llamadas a funciones) transmitidos por la red.
- Cuando un cliente envía una petición para que se realice una operación, decimos que el cliente **invoca una operación** del servidor.

# Principios arquitectónicos

- Extrapolando los principios de diseño modular de programas, podemos enunciar los principios arquitectónicos clave
  - Responsabilidad única
    - Cada módulo debe ser responsable de una funcionalidad específica y concreta
  - Separación de la funcionalidad
    - Dividir la aplicación en bloques funcionales independientes
    - Minimizar los puntos de interacción con un acoplamiento pequeño

# Arquitectura cliente / servidor



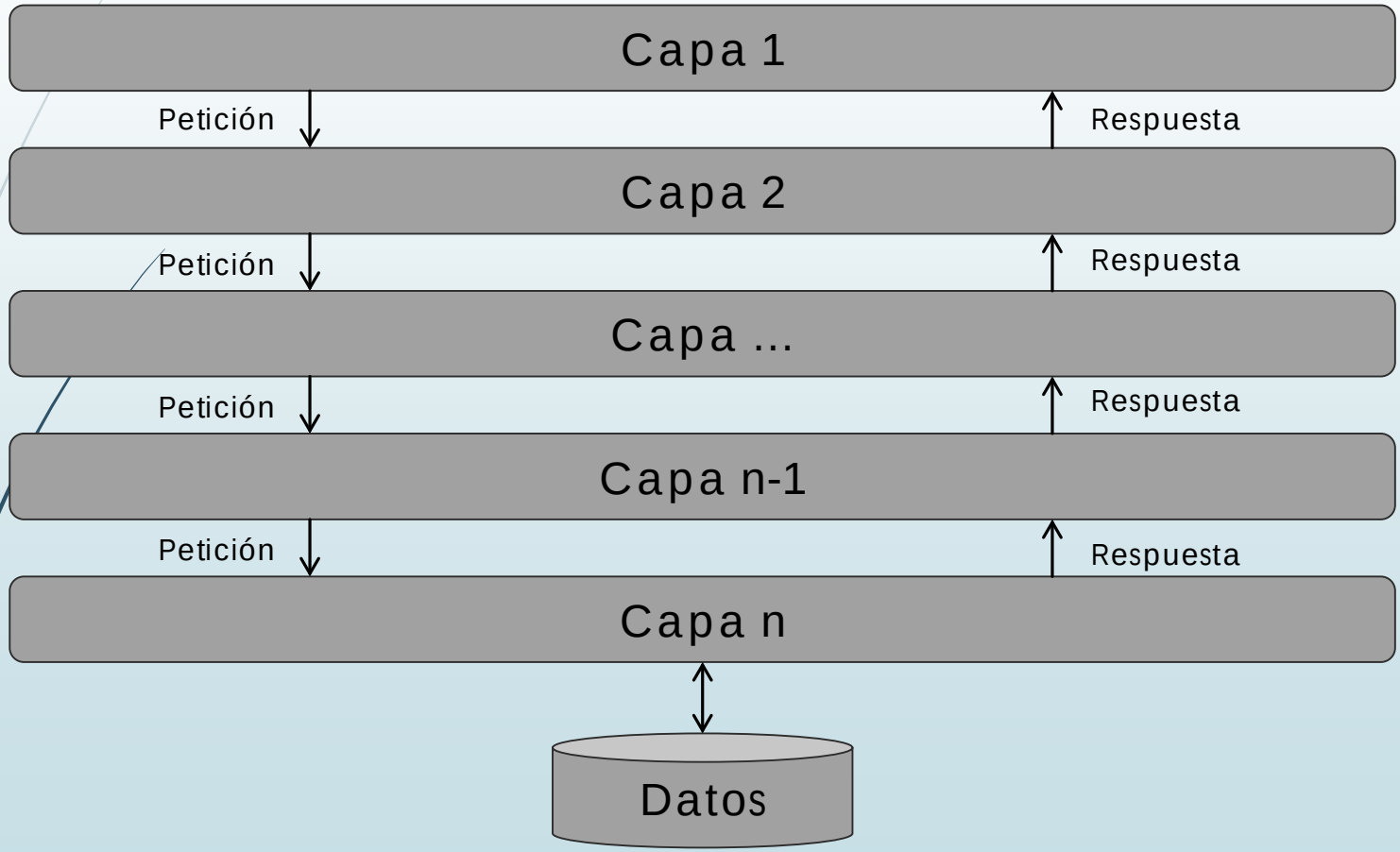
# Arquitectura cliente / servidor

- También denominada arquitectura de 2 capas
- Ejemplos de esta arquitectura:
  - Aplicación de escritorio / servidor de bases de datos
  - Navegador web / servidor web
  - Cliente de correo / servidor de correo
- Ventajas:
  - Acceso centralizado a los datos
    - Mayor facilidad a la hora de administrarlos
  - Incremento de seguridad
    - Sólo es necesario proteger el servidor y la transmisión
  - Facilidad de mantenimiento y escalado
    - La replicación del servidor es transparente a los clientes

# Arquitectura cliente / servidor

- Desventajas
  - Los datos y la lógica del negocio se mezclan en el servidor, lo que afecta a la escalabilidad y la extensibilidad
  - El servidor es un punto único de fallo
- Raramente usada en la actualidad
  - La mezcla de datos y lógica de negocio se resuelve con una arquitectura basada en capas
  - El punto único de fallo del servidor se resuelve con arquitecturas de basadas en capas o arquitecturas orientadas a servicios

# Arquitectura basada en capas



# Arquitectura basada en capas

- Funcionamiento:
  - Cada capa agrupa funcionalidad relacionada
  - Las capas se apilan verticalmente entre los usuarios y los datos
  - La comunicación entre capas es explícita mediante interfaces bien definidos y débilmente acoplada
  - Los componentes de una capa sólo pueden interactuar entre sí o con los de la capa inferior

# Arquitectura basada en capas

- Principios de diseño
  - **Abstracción y encapsulación**
    - Cada capa abstrae toda la funcionalidad que incluye y sólo es necesario entender su interfaz
    - No es necesario preocuparse de los detalles de implementación de la capa ni de sus componentes
  - **Comunicación sencilla**
    - La comunicación es sencilla. Las ordenes viajan hacia abajo por la pila de capas, y las respuestas retornan hacia arriba
  - **Alta cohesión, acoplamiento bajo**
    - Cada capa tiene una funcionalidad clara y acotada
    - El envío de peticiones a capas inferiores se realiza mediante un interfaz claro

# Arquitectura basada en capas

- Ventajas
  - **Abstracción**
    - Cada capa se define de forma abstracta antes de empezar la implementación
    - La pila jerárquica permite aumentar el nivel de abstracción en cada paso
  - **Aislamiento**
    - Las capas son independientes. La tecnología de su implementación puede variar sin afectar a las demás
  - **Manejabilidad**
    - Las dependencias son claras y explícitas, con lo que los cambios son fácilmente localizables
  - **Rendimiento**
    - Las capas se pueden distribuir en varios sistemas para mejorar la escalabilidad, tolerancia a fallos y rendimiento
  - **Reutilización**
    - Las capas inferiores no tienen dependencias con las superiores, permitiendo su uso potencial en otros escenarios
  - **Capacidad de prueba**
    - Los interfaces claros y definidos permiten construir simuladores de las capas

# Arquitectura basada en capas

- Desventajas
  - Problemas de rendimiento debido a la sobrecarga de llamadas entre las capas
  - Aumento en el coste de desarrollo debido a la necesidad de implementar nueva funcionalidad en todas las capas
  - Cambios en los niveles inferiores pueden provocar cambios en los niveles superiores

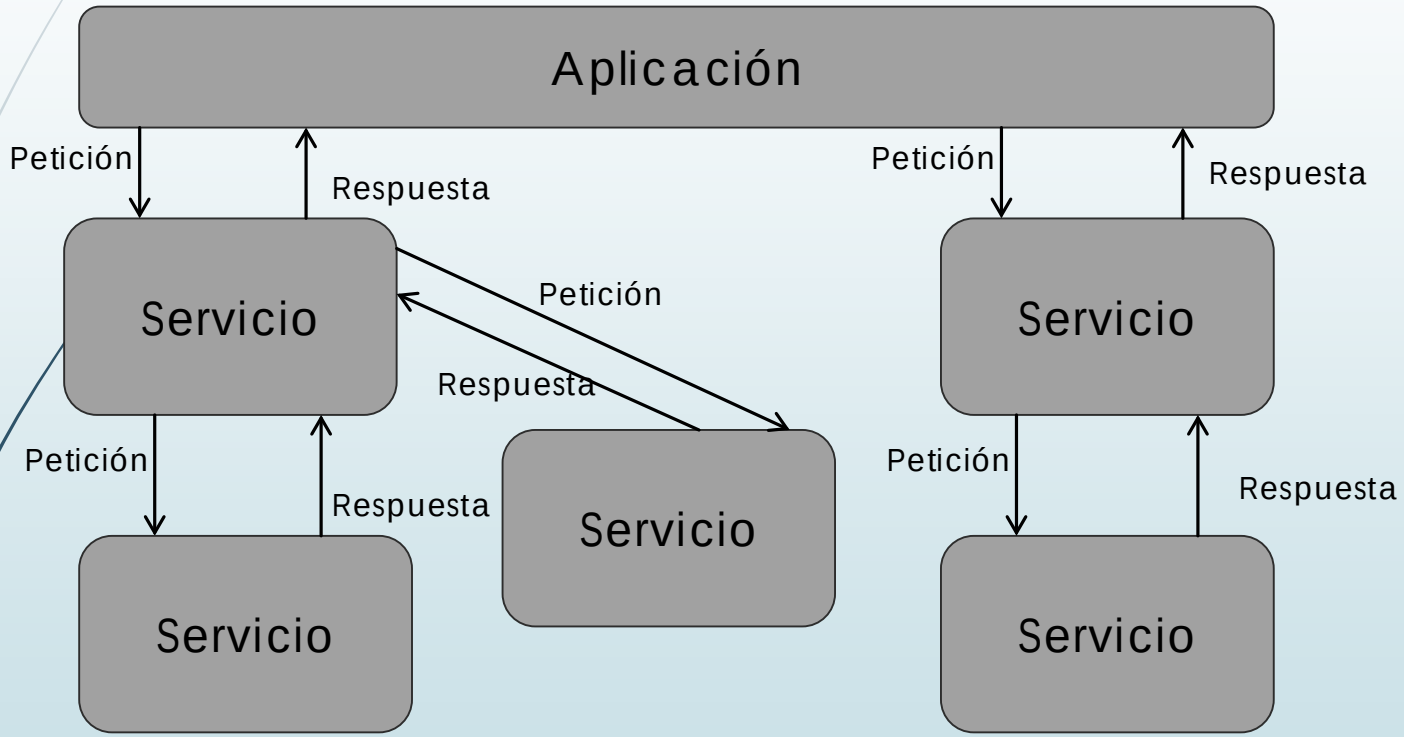
# Arquitectura en 3 capas / n capas

- Funcionamiento
  - Patrón similar a la arquitectura basada en capas
  - Cada capa debe ubicarse en un servidor independiente
  - La jerarquía de capas es rígida. Sólo se conoce la capa inmediatamente inferior
  - Es común considerar 3 capas:
    - Acceso a datos / modelo
    - Lógica de negocio / controlador
    - Presentación al usuario / vista

# Arquitectura en 3 capas / n capas

- Ventajas
  - **Mantenibilidad.** La independencia de las capas hace que los cambios internos no afecten a las otras capas
  - **Escalabilidad y flexibilidad.** Debido a la asignación de capas a servidores
- Inconvenientes
  - Aumenta el tráfico de red, la necesidad de balanceado de carga, y la tolerancia a fallos

# Arq. orientada a servicios



# Arq. orientada a servicios

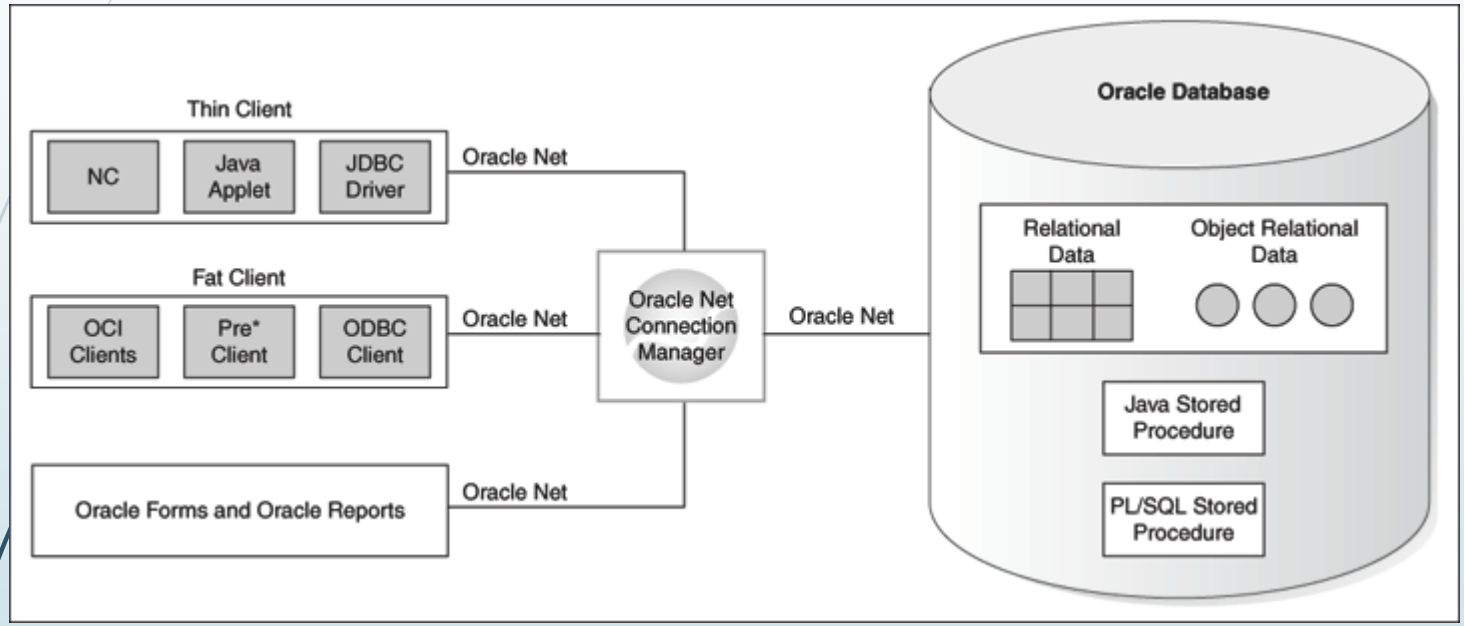
- Funcionamiento
  - La funcionalidad de la aplicación se implementa mediante servicios con estas características
    - **Autónomos.** Se desarrolla y despliega independientemente
    - **Distribuíbles.** Un servicio se ubica en cualquier lugar de la red
    - **Débilmente acoplados.** La dependencia entre servicios es sólo a nivel de interfaz
    - **Políticas definidas.** Existe un contrato de lo que hace el servicio, un esquema de los datos utilizados , y un acuerdo en el protocolo
  - Protocolos para servicios
    - Simple Object Access Protocol (SOAP)
    - XML Remote Procedure Call (XML-RPC)
    - Representationtal State Transfer (REST)

# Arq. orientada a servicios

- Ventajas
  - Modularidad, reusabilidad y abstracción
  - Interoperabilidad. Los protocolos de comunicación son independientes de la tecnología
  - Flexibilidad. El despliegue es variable y los servicios pueden ser encontrados por su contrato

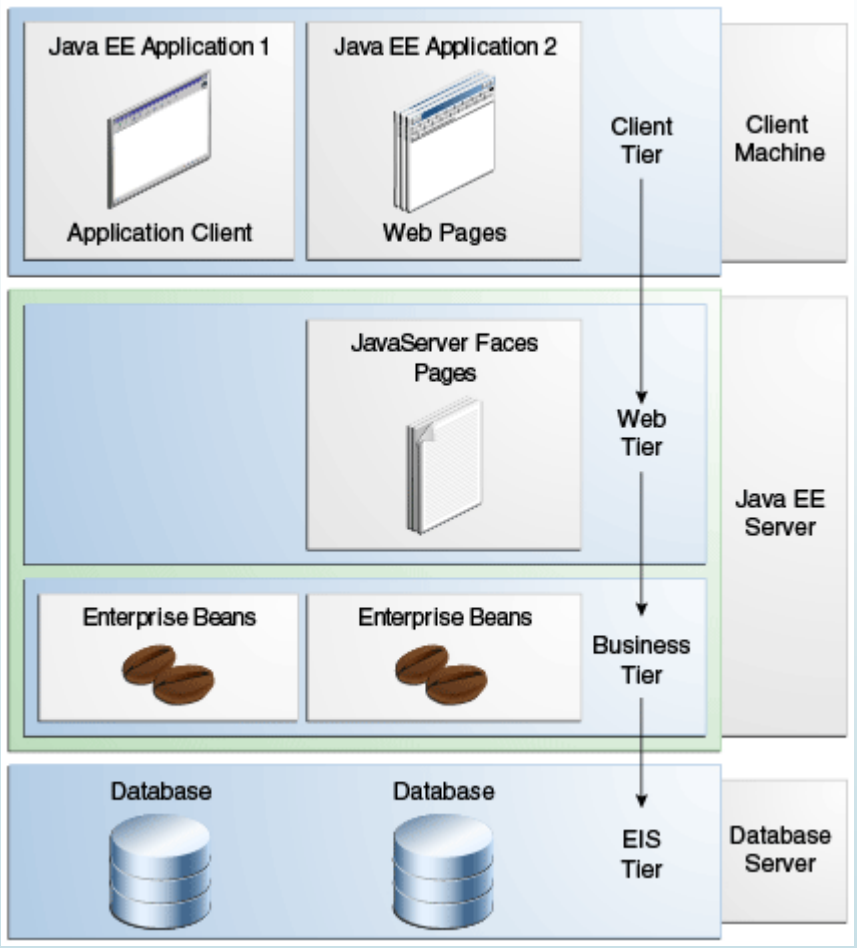
# Ejemplos

# Oracle Two-Tier Client/Server



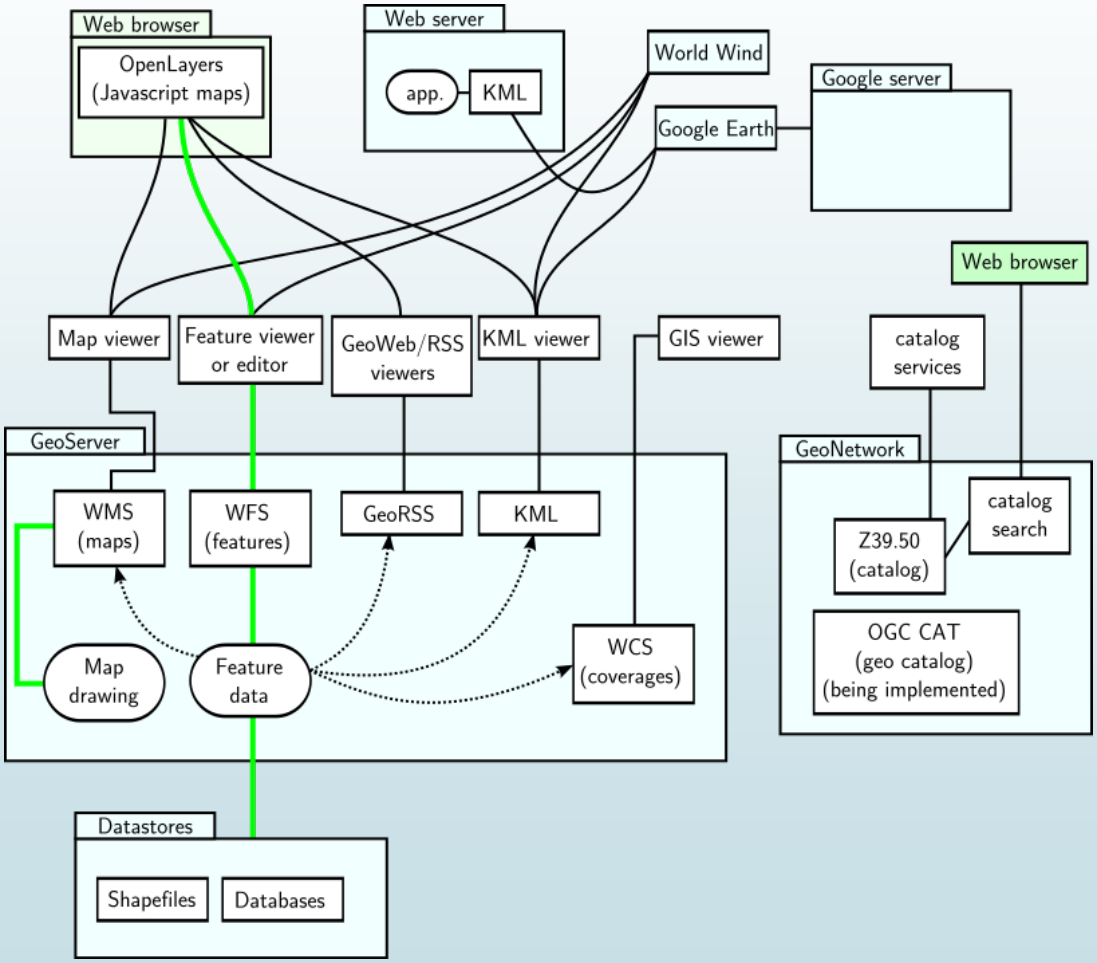

Fuente: http://docs.oracle.com/database/121/JJDEV/chone.htm#JJDEV13021

# Java EE Tutorial




Fuente: http://docs.oracle.com/javaee/7/tutorial/doc/overview003.htm#BNAAY

# Servicios OGC




Fuente: https://en.wikipedia.org/wiki/Open_Geospatial_Consortium