CÁTEDRA

Ingeniería de Software II

2019

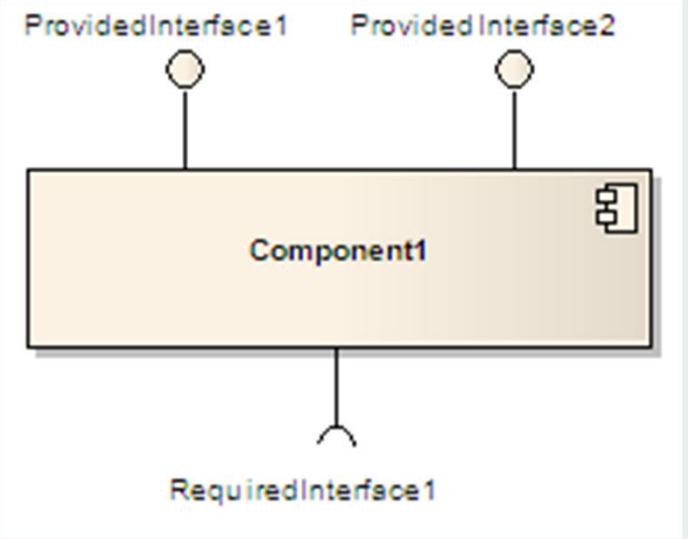
RECORDAMOS

Componente

Es una parte modular de un sistema que encapsula su contenido y cuya manifestación es reemplazable dentro de su entorno

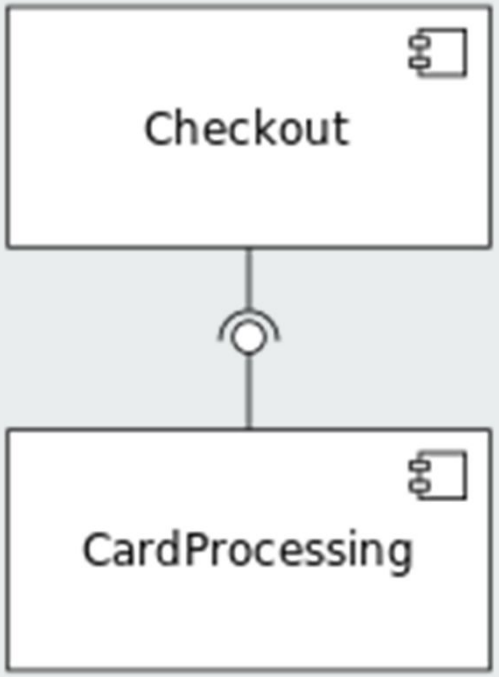
Actúa como una caja negra cuyo comportamiento externo está completamente
definido por sus interfaces proporcionadas y requeridas.

Debido a esto, un componente se puede reemplazar por otro que soporte el
mismo protocolo.



RECORDAMOS

El componente Checkout (comprobación), responsable de facilitar los pedidos del cliente, *requiere* al componente CardProcessing (procesador de tarjetas) para cargar el monto a la tarjeta de crédito/débito del cliente (funcionalidad que este último *provee*).



Diseño Componentes-Interfaz

Primero se definen los componentes - las grandes piezas del sistema - y después las interfaces correspondientes

Una vez que los componentes y las interfaces se han definido es posible dividir la implementación del sistema entre los participantes organizandolos en varios grupos los desarrolladores son libres de implementar los componentes.

Diseño a partir de clases

Utilizando las clases dominio y el método a partir de las clases se tiende más a la resolución del problema

En caso de que la complejidad de la solución se incremente o se identifique un grupo de clases que pueda depurarse y reutilizarse con más facilidad si se les encapsula en componentes.

Estereotipo	Semántica
<<buildComponent>>	Un componente que define un conjunto de elementos para fines organizativos o de desarrollo a nivel de sistema.
<<entity>>	Un componente de información persistente que representa un concepto de negocio.
<<implementation>>	Una definición de componentes que no tiene especificación, es una implementación para una <<specification>> aparte con la que tiene una dependencia.
<<specification>>	Un clasificador que especifica un dominio de objetos sin definir la implementación física de esos objetos, por ejemplo, un componente estereotipado por<<specification>> solamente tiene interfaces proporcionadas y requeridas y no clasificadores de realización.
<<process>>	Un componente basado en transacción.
<<service>>	Un componente funcional sin estado que computa un valor.
<<subsystem>>	Una unidad de descomposición jerárquica para grandes sistemas.

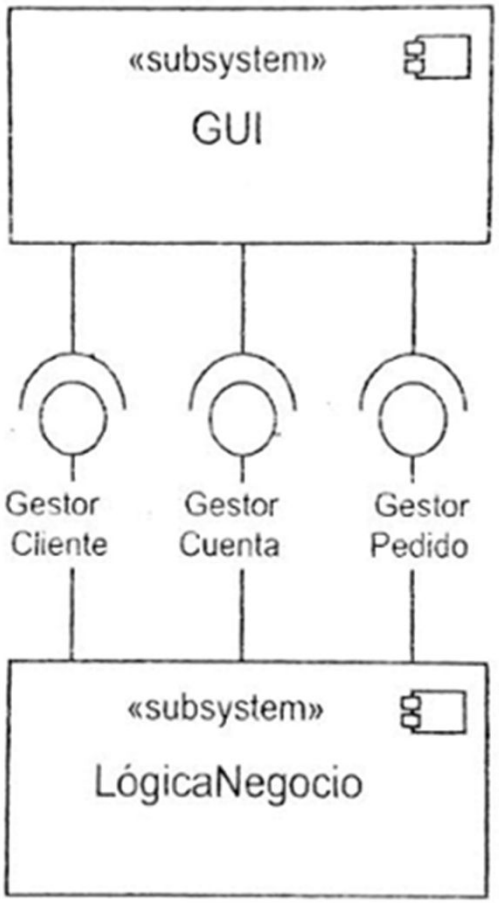
Un subsistema es un componente que actúa como una unidad de descomposición para un sistema mayor

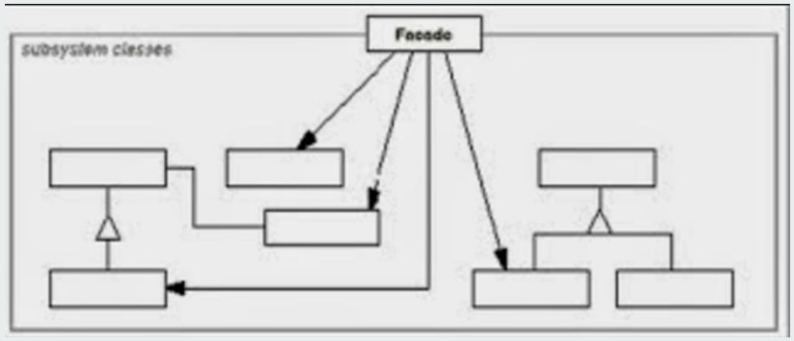
Un subsistema es una construcción lógica que se utiliza para descomponer un sistema grande en bloques manejables.

Desglosar un sistema en subsistemas resuelve un problema de desarrollo difícil en muchos subproblemas más pequeños y manejables

El subsistema GUI solamente sabe de las interfaces GestorCliente, GestorCuenta, y Gestor Pedido. No sabe nada del subsistema de LógicaNegocio que lo implementa.

Esto significa que podría en principio reemplazar el susbsistema LógicaNegocio con otro subsistema siempre y cuando proporcionen las mismas interfaces.





Estructurar un sistema en **subsistemas** ayuda a reducir la complejidad. Un objetivo común de diseño es minimizar la comunicación y dependencias entre subsistemas.
Una forma de conseguir este objetivo es introducir un objeto **fachada** que proporciona una única interfaz simplificada a las facilidades más generales de un subsistema.

UNIDAD 4: Interfaces y componentes

Patrón de diseño Fachada / Facade

Facade: Conoce cuáles clases del subsistema son responsables de una petición y delega las peticiones de los clientes en los objetos del subsistema.
Clases del subsistema: Implementan la funcionalidad del subsistema, manejan el trabajo asignado por el objeto Facade y además de esto no tienen ningún conocimiento del Facade.

Nombre del patrón: Facade o Fachada

Clasificación del Patrón: Estructural

Intención: Proporcionar una interfaz simplificada para un grupo de subsistemas o un sistema complejo.

Motivación: Simplificar el acceso a un conjunto de clases proporcionando una única clase que todos utilizan para comunicarse con dicho conjunto de clases.

Reducir la complejidad y minimizar dependencias

Introducción al ejemplo:

Vamos a realizar software para una inmobiliaria.

Se realizan trabajos diferentes:
como el cobro de alquiler, muestra de inmuebles, administración de consorcios, contratos de ventas, contratos de alquiler, etc.

Por una cuestión de seguir el paradigma de programación orientada a objetos, es probable que no se realice todo a una misma clase, sino que se dividen las responsabilidades en diferentes clases.

Vayamos realizando el diagrama de clases:

Imaginemos que en el soft de la inmobiliaria tenemos diversos tipos de Personas, todas con sus atributos y métodos correspondientes. Acá pondré sólo un resumen de ellas, ya que no tiene sentido ahondar en demasiados detalles de implementación.

```
public abstract class Persona {}
public class Cliente extends Persona {}
public class Interesado extends Persona {}
public class Propietario extends Persona {}
```

Lo mismo con los métodos principales de las diversas clases que tiene el sistema:

```
public class AdministracionAlquiler {
	public void cobro (double monto){
		// algoritmo
	}
}

public class CuentasAPagar {
	public void pagoPropietario(double monto){
		// algoritmo
	}
}

public class MuestraPropiedad {
	public void mostraPropiedad(int numeroPropiedad){
		// algoritmo
	}
}

public class VentaInmueble {
	public void gestionaVenta(){
		// algoritmo
	}
}
```

```java
public class Inmobiliaria {
    private MuestraPropiedad muestraPropiedad;
    private VentaInmueble venta;
    private CuentasAPagar cuentasAPagar;
    private AdministracionAlquiler alquiler;

    public Inmobiliaria() {
        muestraPropiedad = new MuestraPropiedad();
        venta = new VentaInmueble();
        cuentasAPagar = new CuentasAPagar();
        alquiler = new AdministracionAlquiler();
    }

    public void atencionCliente(Cliente c) {
        System.out.println("Atendiendo a un cliente");
    }

    public void atencionPropietario(Propietario p) {
        System.out.println("Atendiendo a un propitario");
    }

    public void atencionInteresado(Interesado i) {
        System.out.println("Atencion a un interesado en una propiedad");
    }

    public void atencion(Persona p) {
        if (p instanceof Cliente) {
            atencionCliente((Cliente) p);
        } else if (p instanceof Propietario) {
            atencionPropietario((Propietario) p);
        } else {
            atencionInteresado((Interesado) p);
```

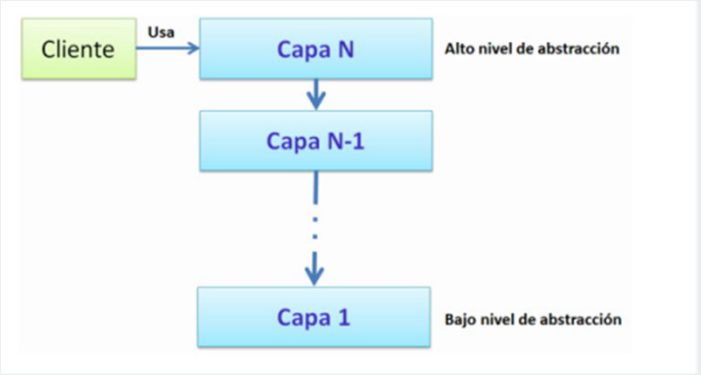
```java
public static void main(String[] args) {
    Cliente c = new Cliente();
    Interesado i = new Interesado();

    // Primer Cliente:
    Inmobiliaria inmo = new Inmobiliaria();
    inmo.atencionCliente(c);
    inmo.atencionInteresado(i);
    MuestraPropiedad muestraPropiedad = new MuestraPropiedad();
    muestraPropiedad.mostraPropiedad(123);
    VentaInmueble venta = new VentaInmueble();
    venta.gestionaVenta();
    AdministracionAlquiler alquiler = new AdministracionAlquiler();
    alquiler.cobro(1200);
    CuentasAPagar cuentasAPagar = new CuentasAPagar();
    cuentasAPagar.pagoPropietario(1100);

    // Segundo Cliente (lo mismo pero usando el Facade)
    Inmobiliaria inmo2 = new Inmobiliaria();
    inmo2.atencion(i);
    inmo2.atencion(c);
    inmo2.mostraPropiedad(123);
    inmo2.gestionaVenta();
    inmo2.cobraAlquiler(1200);
    inmo2.paga(1100);

}
```

El primero de los clientes esta obligado a conocer muchos detalles de los subsistemas y el segundo no.

La colección de subsistemas de diseño e interfaces constituye la arquitectura de alto nivel de un sistema

Para que la arquitectura sea fácil de entender y mantener, necesitamos organizar la colección de subsistemas e interfaces de forma coherente.

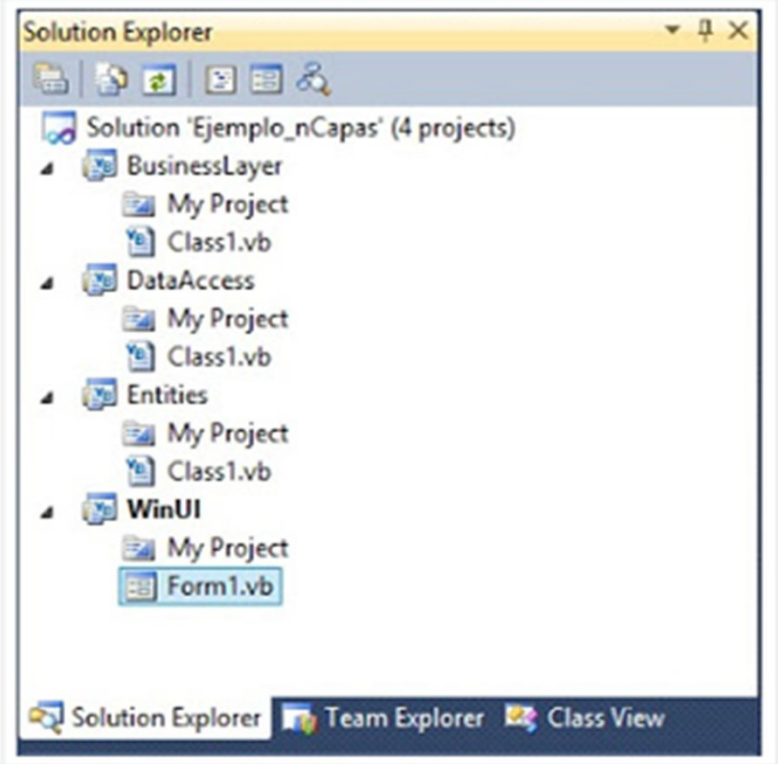
Para esto se aplica un patrón de arquitectura conocido como **disposición en capas**

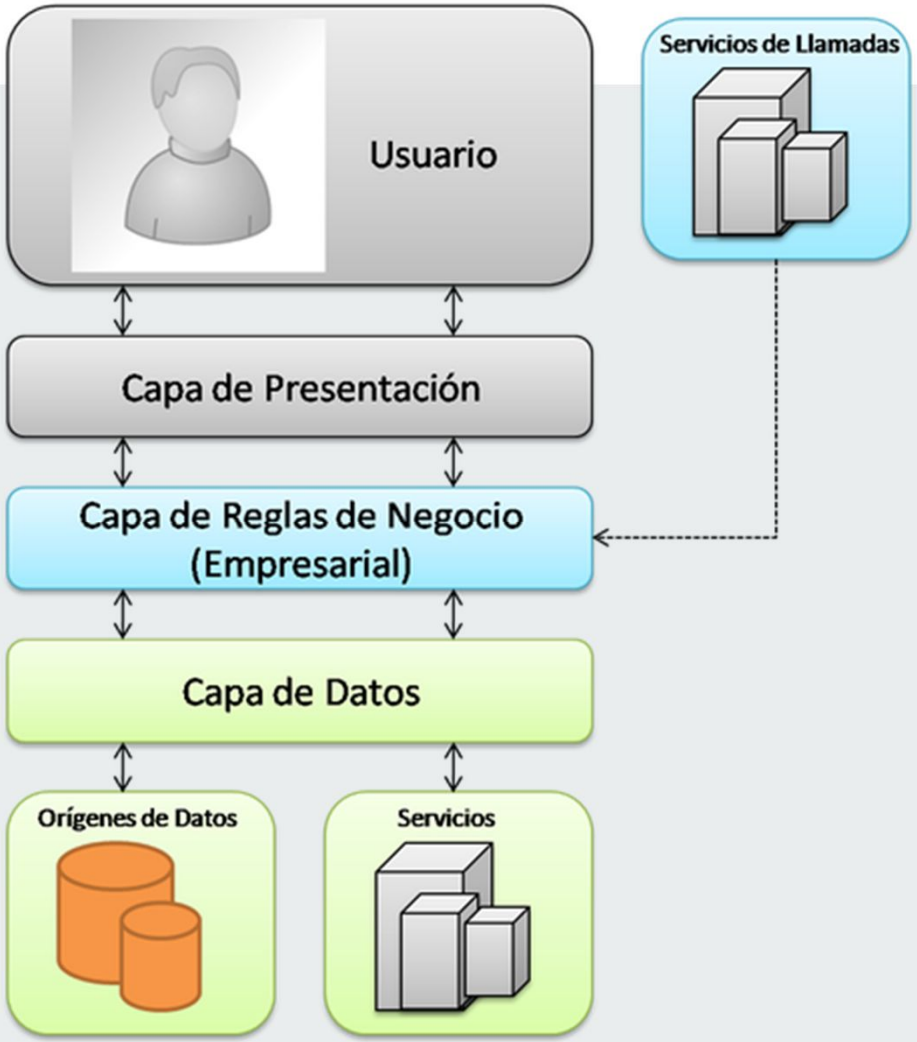


El patrón en Capas se utilizar para descomponer un problema en subtareas…. un sistema de gran tamaño que requiere la descomposición.

Estructura tu sistema en un número apropiado de capas. Comience en el nivel más bajo de abstracción sistema. Trabaja la abstracción poniendo la capa J en la parte superior de la capa de J-1 hasta llegar al nivel superior de la funcionalidad, llámalo capa N. La principal característica estructural del patrón de capas es que los servicios de la capa J sólo son utilizados dependencias responsabilidad de la capa J es proveer servicios usados por la capa J+1 y delegar sub tareas a la capa J-1.

UNIDAD 4: Interfaces y componentes

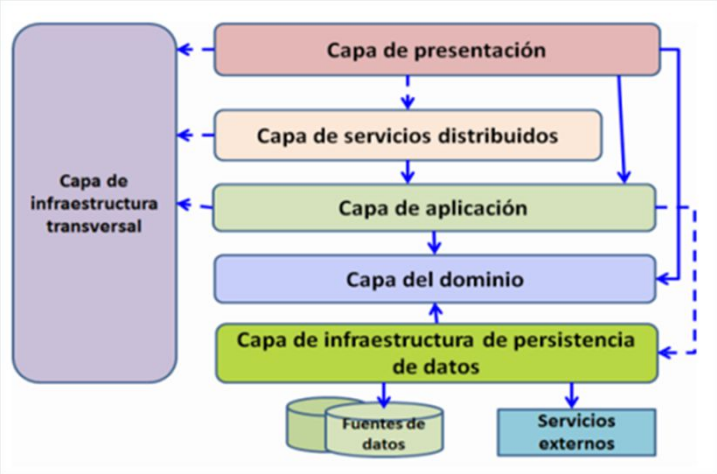




Existen muchas formas de producir arquitecturas en capas y puede tener tantas capas como tenga sentido.

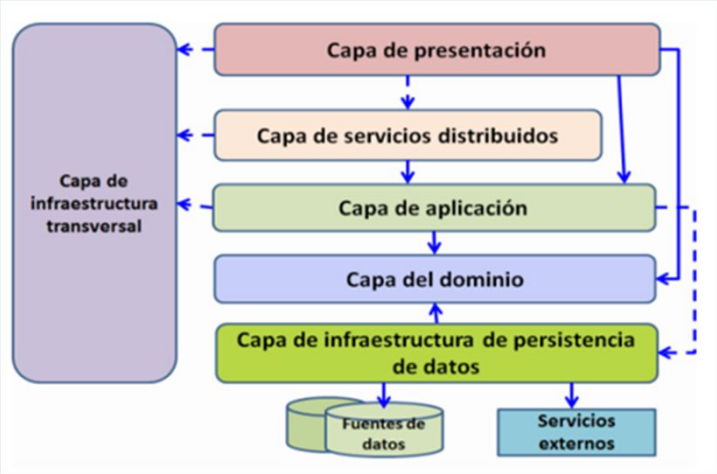
El patrón básico es una división entre las capas de presentación, la lógica de negocio y las capas de utilidad.

Es bastante común subdividir la capa lógica de negocio.



Patrón N-Capas con Orientación al Dominio

Presentación: responsable de mostrar información e interpretar acciones
Servicios distribuidos: Cuando una aplicación actúa como proveedor de servicios para otras aplicaciones remotas, o incluso también localizaciones pública negocio servicios. (habitualmente Servicios Web) proporciona un medio de acceso remoto basado en canales de comunicación y mensajes de datos.
Aplicación: Define los trabajos que la aplicación como tal debe de realizar y redirige a los objetos del dominio y de infraestructura (persistencia, etc.) que son los que internamente deben resolver los problemas. Sirve principalmente para coordinar la lógica del flujo del caso de uso.



Patrón N-Capas con Orientación al Dominio

Dominio: Esta capa es responsable de representar conceptos de negocio e implementación de las reglas de dominio. Estos componentes implementan la funcionalidad principal del sistema y encapsulan toda la lógica de negocio relevante

Persistencia de datos: proporciona la capacidad de persistir datos asi como logicamente acceder a ellos. Pueden ser datos propios o datos expuestos por otros (web services)

Ahora uds!!!

Lunes 13: presentación por equipo del tema

Realización de Casos de uso: diseño

Explicación y ejemplos

Como base tomar la unidad 20

UNIDAD 4: Interfaces y componentes

Fuente:

UML 2

- “19 Interfaces y componentes”
- ”20 Realización de Casos de uso: diseño”