



Tecnologías de Programación

Paradigma Orientado a Objetos

III – Reglas del buen diseño



Reglas del Buen Diseño

- COHESION - ACOPLAMIENTO
- BREAK/CONTINUE
- UNICO PUNTO DE SALIDA
- LEY DE DEMETER
- TELL DON'T ASK (TDA)

Toda *ley* o *directriz* de diseño debería ser eso una directriz.
Siempre hay excepciones a estas reglas.

El arte es saber cuando saltárselas :)



Cohesión

La cohesión tiene que ver con la forma en la que agrupamos unidades de software en una unidad mayor. Por ejemplo, la forma en la que agrupamos funciones en una librería, o la forma en la que agrupamos métodos en una clase, o la forma en la que agrupamos clases en una librería, etc...

Se suele decir que cuanto más cohesionados estén los elementos agrupados, mejor.

Acoplamiento

El término "**acoplamiento**" hace alusión al grado de dependencia que tienen dos unidades de software.

¿Qué es una unidad de software? Cualquier pieza de software que realice algún cometido. Por ejemplo: una función, un método, una clase, una librería, una aplicación, un componente, etc...



Utilización de Break/Continue en ciclos

```
Integer i = 0;
// Un ciclo "infinito":
while(true) {
    i++;
    Integer j = i * 27;
    if(j == 1269) break; // Fuera del Ciclo
    if(i % 10 != 0) continue; // Regreso al Inicio del Ciclo
    System.out.println(i);
}
```



Único punto de salida

```
//Este método devuelve el doble de a
//si a es par, y devuelve a tal cual si no lo es.

public Integer prueba(Integer a) {
    if (a % 2 == 0)
        return a * 2;
    else
        return a;
}
```



Ley de Demeter

La conocida como ***Ley de Demeter*** o ***del buen estilo***, nos garantiza, durante un desarrollo orientado a objetos una buena escalabilidad, depuración de errores y mantenimiento, ya que ayuda a ***maximizar la encapsulación***. Esto ayuda a mantener un nivel bajo de acoplamiento. Es una norma muy simple de seguir.

A menudo, el contenido de la ley se abrevia sólo con una frase:

“Habla sólo con tus amigos”



Demeter II

Un **método M** de un **objeto O** solo debería invocar métodos:

- suyos
- de sus parámetros
- objetos que cree o instancie
- objetos miembros de la clase (atributos)



Demeter III

Una forma de comprender mejor esta ley podemos dar la vuelta al enunciado y enumerar los casos prohibidos: **no se debe llamar a métodos de los objetos devueltos por otros métodos.**

El caso más común que debemos evitar son las cadenas de métodos, de la forma:

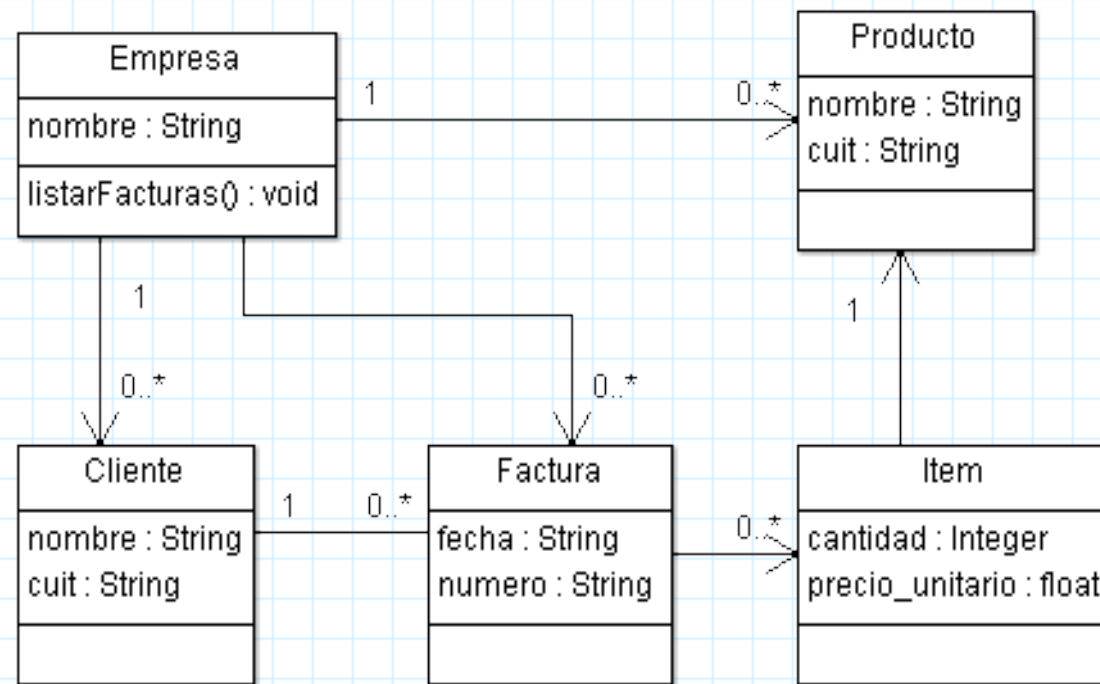
a.getX().getY().getValue();

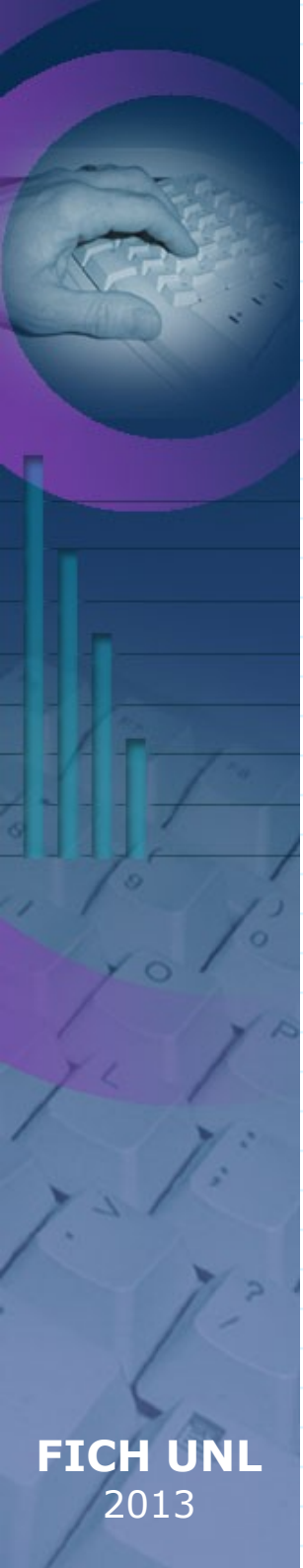
y sustituirlas por funciones que realicen dicha acción:

a.getXYValue();

Demeter - Ejemplo

Tomamos el ejercicio 1 del TP 13.





```
public class Empresa {

    private Collection<Producto> cProductos;
    private Collection<Factura> cFacturas;
    private Collection<Cliente> cClientes;

    public void listarTotalFacturas() {

        for(Factura oFactura : cFacturas) {
            if(oFactura.estaAnulada()) {
                Float totalFactura = 0;
                Collection<Item> cItems = oFactura.publicarItems();
                for (Item oItem : cItems) {
                    totalFactura += oItem.calcularTotalItem();
                }
                System.out.println("Total de la factura " + totalFactura);
            }
        }
    }

    for(Factura oFactura : cFacturas) {
        if(oFactura.estaAnulada()) {
            float totalFactura = oFactura.calcularTotal();
            System.out.println("Total de la factura " + totalFactura);
        }
    }
}
```



Tell don't ask - TDA (Dile, no le preguntes)

Esta es una Regla del buen Diseño más restrictiva que Demeter.

Establece que en toda comunicación entre objetos, el emisor del mensaje debe decirle **que hacer** al receptor, no **pedirle** algún atributo, tomar una decisión y a continuación decirle que hacer.

Pasando en limpio, el único que puede tomar una decisión en base al estado de alguno de sus atributos es el objeto que posee dichos atributos.

Un objeto no puede tomar una decisión en base al estado de un atributo que no es suyo.



TDA

¿Cuándo está permitido pedirle un valor a un objeto ?

- Se le puede pedir un valor de estado a un objeto (get) siempre que el dato no se utilice para tomar una decisión.

// Caso incorrecto -- tomo una decisión en base al estado interno de otro objeto.

```
for(Factura oFactura : cFacturas) {  
    if(oFactura.estaAnulada()) {  
        Float totalFactura = oFactura.calcularTotal();  
        System.out.println("Total de la factura " + totalFactura);  
    }  
}
```

// Caso correcto

```
for(Factura oFactura : cFacturas) {  
    System.out.println("Total de la factura " + oFactura.calcularTotal());  
}
```