Tecnologías de Programación

Paradigma Orientado a Objetos

III – Reglas del buen diseño

Ing. Pablo Garello

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. El criterio por el cual se agrupan es la cohesión.

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...

Cuando dos unidades de software son absolutamente independientes (*cada una puede hacer su trabajo sin contar para nada con la otra*), encontramos el grado más bajo de acoplamiento, y decimos que ambas unidades están totalmente desacopladas.

Cohesión y Acoplamiento

Tener unos buenos criterios para agrupar unidades de software (***alta cohesión***), y mantener esas unidades lo más independientes posible (***bajo acoplamiento***) garantiza la modularidad, facilitando la reutilización del software y gran parte de las tareas del desarrollo del sofware.

Utilización de Break/Continue en cliclos

```
int i = 0;
// Un ciclo "infinito":
while(true) {
    i++;
    int 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 int prueba(int 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:

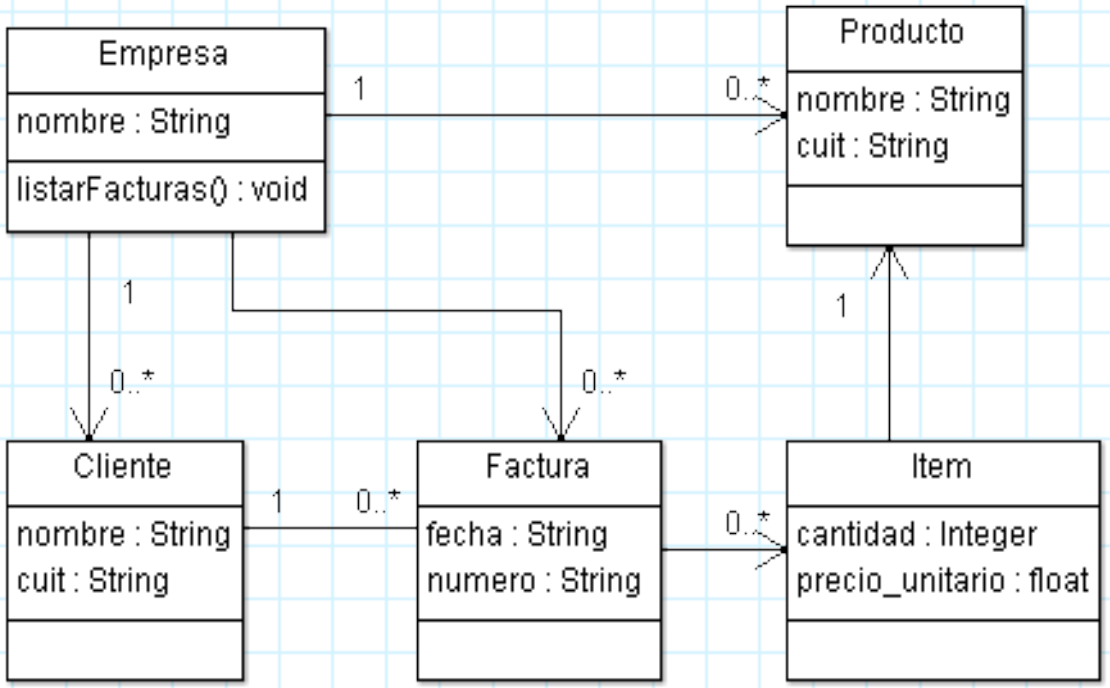
a.getX().getY().getValue();

y sustituirlas por funciones que realicen dicha acción:

a.getXYValue();

Demeter - Ejemplo

Tomamos el ejercicio 2 del TP 13.



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 o dato. 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.