1. Los objetos colaboran

En POO lo mejor ir encontrando relaciones a partir del comportamiento esperado en los escenarios.

Lo más importante en POO no es tanto la estructura de relacionamiento de los objetos, sino el comportamiento. Por lo tanto, deberíamos ocuparnos de, siguiendo el diseño por contrato, determinar los contratos de cada objeto a implementar (precondicionesa y postcondiciones, invariantes, etc.)

1.1. Dependencia y asociación

Definición 1.1. Dependencia Un objeto depende de otro cuando debe conocerlo para poder enviarle un mensaje. Corolario: Todo objeto cliente depende de su servidor.

Esta dependencia puede venir dada de tres maneras:

- Porque el objeto servidor se envía como argumento.
- Porque el objeto servidor se obtiene como respuesta al envío de un mensaje a otro objeto.
- Porque el objeto cliente tiene una referencia al servidor. (Esto es asociación)

Definición 1.2. Asociación Una asociación es una forma de dependencia en la que el objeto cliente tiene almacenada una referencia al objeto servidor.

Tanto las dependencias más débiles como las asociaciones tienen un sentido: decimos que el cliente depende del servidor, y no al revés. Para un mismo mensaje, lo habitual es que las dependencias sean unidireccionales.

Ejemplo 1.1. Ejemplos de dependencia y asociación (preguntar si es correcto)

- Si tengo una clase A donde los atributos usan una clase B. Esto sería asociación por que el objeto cliente tiene almacenado una referencia al objeto servidor?
- Si tengo una clase A donde sus métodos usan una clase B. Esto sería dependencia por que el objeto cliente, depende objeto servidor?

Sí, tienes razón en ambos casos.

Si tienes una clase A donde los atributos usan una clase B, esto sería una asociación porque el objeto cliente (clase A) tiene almacenada una referencia al objeto servidor (clase B). Esto significa que el objeto cliente puede interactuar con el objeto servidor a través de esta referencia.

Si tienes una clase A donde sus métodos usan una clase B, esto sería una dependencia porque el objeto cliente (clase A) depende del objeto servidor (clase B). Esto significa que cualquier cambio en la definición de la clase B podría afectar a la clase A. Sin embargo, esta dependencia puede ser débil si se hace que el código dependa de interfaces o clases abstractas en lugar de clases concretas.

Definición 1.3. (Digresión) Conceptualmente, si una prueba está probando el comportamiento de más de una clase, o incluso más de una responsabilidad de un objeto, es una prueba de integración, no una prueba unitaria.

1.2. Programación por diferencia: herencia

Definición 1.4. Herencia La herencia es una relación entre clases, por la cual se define que una clase puede ser un caso particular de otra. A la clase más general la llamamos madre y a la más patricular hija.

Corolario: Cuando hay herencia, todas las instancias de la clase hija son también instancias de la clase madre.

Definición 1.5. programación por diferencia Programamos por diferencia cuando indicamos que parte de la implementación de un objeto está definida en otro objeto, y por lo tanto sólo implementamos las diferencias específicas.

Como regla, entonces, conviene hacer siempre este test a una relación para ver si aplicar o no la herencia: *¿necesitamos reutilizar la interfaz de una clase tal como está en otra clase, sin que nada me sobre?* Si es así, puedo usar herencia, haciendo que la clase que va a reutilizar sea hija de la clase que provee el código que nos interesa. Si no, conviene reutilizar por delegación.

Dejar la herencia solamente para aquellos casos en que la clase madre tenga una interfaz contenida en la interfaz de la clase hija. Es decir, que la clase madre no tenga métodos que sobren en la clase hija.