Relaciones entre clases en Java

Ricardo Pérez López

IES Doñana, curso 2020/2021

Generado el 14 de abril de 2021 a las 18:09:00

Índice general

1. Asociaciones básicas

1.1. Agregación

La **agregación** se consigue haciendo que el objeto agregador contenga, entre sus variables de instancia, una referencia al objeto agregado.

Para que sea agregación, la vida del objeto agregado **no** debe depender necesariamente del objeto agregador; es decir, que al destruirse el objeto agregador, eso no signifique que se tenga que destruir también al objeto agregado.

Eso implica que puede haber en el programa varias referencias al objeto agregado, no sólo la que almacena el agregador.

Lo habitual en la agregación es que la variable de instancia que almacene la referencia al objeto agregado se asigne, o bien directamente (si la variable tiene la suficiente visibilidad) o bien a través de un método que reciba la referencia y se la asigne a la variable de instancia.

Ese método puede ser (y suele ser) un constructor de la clase.

Ejemplo:

```
class Agregador {
    private Agregado ag;

    public Agregador(Agregado ag) {
        setAg(ag);
    }

    public Agregado getAg() {
        return ag;
    }

    public void setAg(Agregado ag) {
        this.ag = ag;
    }
}
```

En la agregación, es frecuente encontrarnos con métodos *getter* y *setter* para la variable de instancia que hace referencia al agregado.

1.2. Composición

La **composición** se consigue haciendo que el objeto compuesto contenga, entre sus variables de instancia, una referencia al objeto componente.

Para que sea composición, la vida del objeto componente **debe depender** necesariamente del objeto compuesto; es decir, que al destruirse el objeto compuesto, se debe destruir también al objeto componente.

Eso implica que sólo puede haber en el programa una sola referencia al objeto componente, que es la que almacena el objeto compuesto.

Por eso, lo habitual en la composición es que el objeto compuesto sea el responsable de crear al objeto componente.

Normalmente, no hay *setters* para el componente y, en caso de haber *getters*, deberían devolver una copia del objeto componente, y no el objeto componente original.

2. Generalización

2.1. Declaración

Java es un lenguaje con generalización simple, por lo que una clase sólo puede ser subclase directa de una única clase.

La relación de generalización directa entre dos clases se declara en la propia definición de la subclase usando la cláusula **`extends`**:

⟨*clase*⟩ ::= [**`public`**] [**`abstract`** | **`final`**] **`class`** ⟨*subclase*⟩ **`extends`** ⟨*superclase*⟩ **`{`**
⟨*miembro*⟩*
`}`

donde ⟨*subclase*⟩ y ⟨*superclase*⟩ son los nombres de la subclase directa y la superclase directa, respectivamente.

Cuando no se especifica la superclase directa a la hora de definir una clase, el compilador sobreentiende que esa clase es subclase directa de la clase `Object`.

2.2. Subtipado entre tipos referencia

A partir de ese momento, se introduce en el sistema de tipos una regla que dice que:

⟨*subclase*⟩ $<_1$ ⟨*superclase*⟩

Por tanto, se puede decir que la subclase es un subtipo de la superclase.

2.3. Herencia

2.4. La clase `Object`

2.5. Visibilidad protegida

3. Polimorfismo

3.1. El principio de sustitución de Liskov

3.2. Conversiones entre tipos referencia

3.2.1. *Widening*

3.2.2. *Narrowing*

3.3. Sobreescritura de métodos

3.3.1. Sobreescritura y visibilidad

3.3.2. `super`

3.3.3. Covarianza en el tipo de retorno