Revisando la Orientación a Objetos: encapsulación de Java

Maurício Aniche
17/03/2021

Hagamos una apuesta. Estoy seguro de que, cuando vea la clase a continuación, verá un problema con ella:

```
class Pedido {
  public String comprador;
  public double valorTotal;
  // otros atributos
}
```

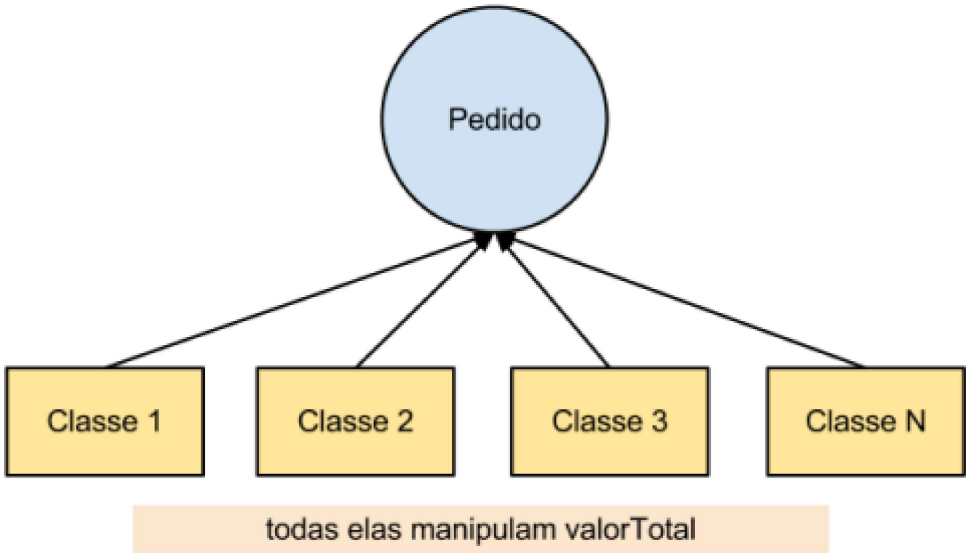
Si. ¡Los atributos son todos públicos! Esto va exactamente en contra de una de nuestras primeras lecciones cuando aprendimos Java: los atributos deben ser privados y necesitamos de getters y setters para accederlos. Así que hagamos este cambio en el

```
  public void setValorTotal(double valorTotal) { this.valorTotal = ValorTotal

  // otros getters y setters
}
```

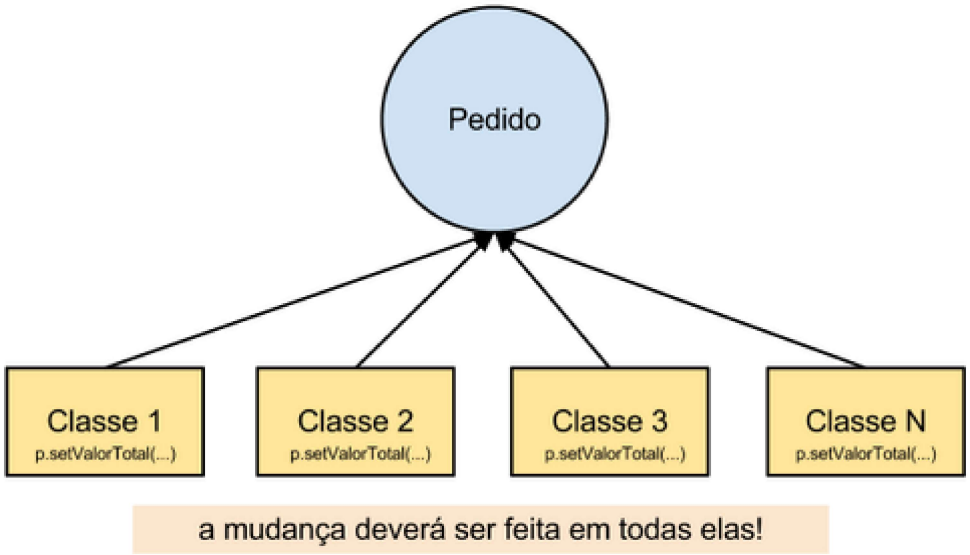
Es mejor ahora, ¿verdad? Todavía no. De hecho, perdemos el gran principio detrás de la idea de colocar atributos como privados. La forma en que la `claseOrdenes` en ese momento, podemos hacer cosas como:

```
Pedido p = new Pedido();
// muda valor do pedido para 200 reais!
p.setValorTotal(p.getValorTotal() + 200.0);
```

Pero, ¿dónde está el problema? Imagínese otras 10 clases que hacen lo mismo: de alguna manera, manejan el monto total del pedido.



Leyenda: Pedido Clase 1 Clase 2 Clase 3 Clase N Todas manipulan valor Total



Leyenda: Pedido Clase 1 Clase 2 Clase 3 Clase N ¡Se deberá hacer el cambio en todas!

¿Cuánto tiempo tardaremos para cambiar el sistema? No sabemos exactamente dónde realizar los cambios, ya que se distribuyen por el código. Este, por cierto, es uno de los grandes problemas de códigos legados: es necesario hacer un cambio simple en tantas clases y, en la práctica, siempre olvidamos algún punto y nuestro sistema a menudo falla.

La clase Pedido no fue bien diseñada. Dimos acceso directo al atributo `valorTotal`, un atributo importante de la clase. Tenga en cuenta que el modificador private en este caso no sirvió de nada, ya que también le dimos un setter. Intentaremos disminuir el acceso al atributo, creando métodos más claros para la operación de depósito:

```
class Pedido {
  private String comprador;
  private double valorTotal;
  // outros atributos

  public String getComprador() { return comprador; }
```

```
Pedido p = new Pedido();
p.agrega(new Item("Ducha Eléctrica", 500.0));
```

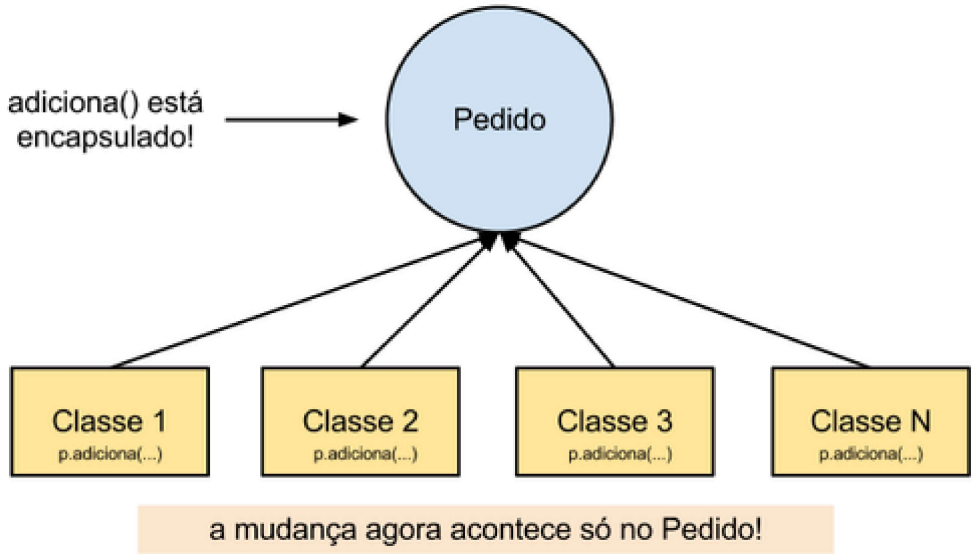
Pero, ¿cuál es la diferencia entre los dos códigos siguientes?

```
Item item = new Item("Super Refrigerador", 1500.0);

// antiguo
if (item.getValor() > 1000) {
    c1.setValorTotal(c1.getValorTotal() + item.getValor() * 0.95);
}
else {
    c1.setValorTotal(c1.getValorTotal() + item.getValor());
}

// nuevo
c1.agrega(item);
```

Veamos que en la primera línea de código, sabemos exactamente **CÓMO** funciona agregar un nuevo item al pedido: debemos tomar el valor total y agregar el nuevo valor con un 5% de descuento si es mayor que 1000. En la segunda línea de código, no sabemos **cómo** funciona este proceso.

Cuando sabemos **QUÉ** hace un método (igual que el método agrega, sabemos que agrega un item al pedido, debido a su nombre), pero no sabemos exactamente cómo lo hace, ¡decimos que este comportamiento es **encapsulado**!

Desde el momento en que las otras clases no saben **cómo** la clase principal hace su trabajo, ¡significa que los cambios sólo se llevarán a cabo en un lugar! Después de todo,



Leyenda: ¡agrega () está encapsulado! Pedido Clase 1 Clase 2 Clase 3 Clase N ¡el cambio ahora ocurre sólo en el Pedido!

Es decir, para implementar la nueva regla de negocios, bastaría con modificar un solo lugar:

```
public void agrega(Item item) {
    if (item.getValor() > 1000) this.valorTotal += item.getValor();
    else this.valorTotal += item.getValor() * 0.95;

    // nueva regla de negocio aquí

  }
```

Al final, la verdadera utilidad de `private` es ocultar el acceso de atributos a los que se debe acceder de forma más inteligente. Pero mira que es inútil poner todos los atributos como private y crear getters y setters para todos. Dejamos que la encapsulación "se filtre" de la misma manera

Puedes leer también:

- Cómo separar palabras de String en Java
- Tomando partes de texto en Java
- Diferencia entre int e Integer en Java