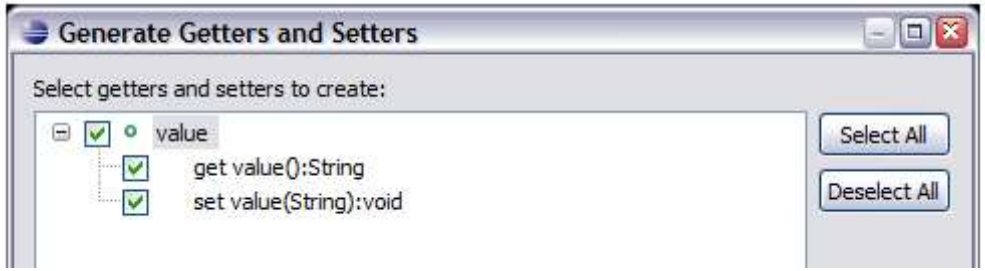
Cómo no aprender Java y Orientación a Objetos: getters y setters

Paulo Silveira

31/03/2021

Mucha gente pregunta:**¿Cómo aprender orientación a objetos?** Hay varias formas de aprender O.O., no creo que haya una mejor, pero hay formas de **no** aprender. Esta publicación, creada en 2006 y actualizada en 2017, muestra cómo el diseño de clases continúa siguiendo algunas importantes reglas de buenas prácticas.



Una de las prácticas más controvertidas que aprendimos al comienzo del aprendizaje de

```
class Cuenta {
    private double limite;
    private double saldo;

    public double getSaldo() {
    return saldo;
    }

    public void setSaldo(double saldo) {
    this.saldo = saldo;
    }

    public double getLimite() {
    return limite;
    }

    public void setLimite(double limite) {
    this.limite = limite;
    }
}
```

¿Cuál es el significado de este código? ¿Para qué este setSaldo? ¿y este setLimite? ¿y el getLimite? ¿Utilizará estos métodos? **Nunca cree un getter o setter sin sentir una necesidad real de ello.**

Esta es una regla para cualquier método, pero particularmente los getters y setters son campeones: muchos de ellos **nunca** se invocarán, y gran parte del resto podría y debería ser reemplazado por métodos de negocios.

```
    private double limite;

    public Cuenta(double limite) {
    this.limite = limite;
    }

    public void deposita (double x) {
    this.saldo += x;
    }

    public void saca(double x) {
    if(this.saldo + this.limite >= x) {
    this.saldo -= x;
    }
    else throw new IllegalArgumentException("¡sobrepasó el límite!");
    }

    public double getSaldo() {
    return this.saldo;
    }
}
```

¡Y ni siquiera estamos hablando de *test driven development*! Probando la clase Cuenta notarás rápidamente que algunos de los getters y setters anteriores no tienen ningún uso y echaría de menos algunos métodos más enfocados en la lógica de negocio de tu aplicación, como el saca y el deposita.

Este ejemplo es muy trivial, pero puedes encontrar muchas clases que no parecen un *java bean* que exponen atributos como Connection, Thread, etc., sin ninguna necesidad.

puñado de atributos! ¿Dónde está la orientación a objetos? Muchas clases de títeres se generan cuando hacemos Value Objects, entidades de *hibernate*, entre otros.*Pero, ¿qué poner en mis entidades de hibernate además de getters y setters?*

Primero que nada, asegúrate si **realmente necesitas** de estos getters y setters.

¿Para qué un *setID* en su clave primaria si su framework usará la reflection o manipulación de bytecode para obtener el atributo privado, o si puedes pasarla por el constructor? Acerca de value objects, ¿realmente necesitas de tus setters? En muchos casos, se crean VO solo para exponer los datos, y no hay ninguna necesidad para los setters... ¡todo lo que se necesita es un buen constructor! De esa manera evitamos un modelo de dominio anémico.

En hibernate suelo utilizar algunos métodos de negocio, algo así como en la clase Cuenta. Mis entidades tienen una cierta responsabilidad, especialmente las relacionadas con los atributos que les pertenecen. Para aquellos que todavía están aprendiendo,está el artículo de POO de Alura, que tiene algunas de estas discusiones y busca enseñar OO comentando siempre estas malas prácticas, como el uso innecesario de la herencia.

Vale la pena recordar que, el ejemplo es solo didáctico. Trabajar con dinero requiere innumerables precauciones, como no usar double. El enfoque de esta publicación es que te detengas y pienses antes de crear el getter y setter. Y, por supuesto, puede ser que su caso justifique la creación de estos métodos.

Vemos mucho estos temas en los cursos de Java en Alura.

Puedes leer también:

- Revisando la Orientación a Objetos: encapsulación de Java
- Redondeo de números en Java
- Conozca la API de fechas de Java 8

PREGUNTAS FRECUENTES

SÍGUENOS EN NUESTRAS REDES SOCIALES

¡CONTÁCTANOS!

¿Dudas, críticas, sugerencias o elogios?

¡Quiero entrar en contacto!

alura LATAM

POWERED BY

grupo alura

AOVS Sistemas de Informática S.A
CNPJ 05.555.382/0001-33