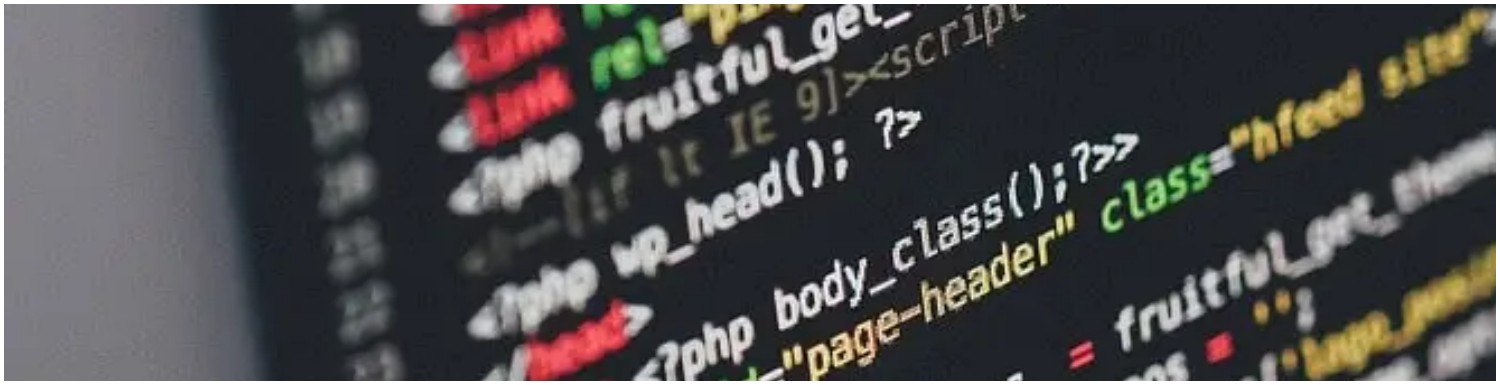

CIENCIA / COMPUTACIÓN

Tell, Don't Ask

por Jose Ramon Pascual | Publicada 8 mayo, 2019

El principio Tell, Don't Ask es una tautología sobre la Programación Orientada a Objetos. Una redundancia del propio concepto de la orientación a objetos en forma de frase que debemos recordar a la hora de desarrollar.

Es fácil dejarse llevar por los dedos y el café, y acabar implementando parte de la lógica que debería encerrar un objeto, fuera de él.

Esto añadirá un problema a nuestro código que afectará a la **reutilización** y al **mantenimiento**: el **acoplamiento**.

El sentido de la Orientación a Objetos es que, propiedades y acciones, estén encapsulados en el propio objeto. Es lo que define una clase.

Si precisamente la esencia de la Programación Orientada a Objetos es definir este encapsulamiento en el desarrollo mas cercano a como vemos las cosas en la vida real ¿Por qué luego tendemos a alejarnos de esta **abstracción**?

Tell, Don't Ask nos recuerda que en POO se debe delegar la responsabilidad de las acciones en el objeto que debe contenerlas. Es decir, devolver la lógica al objeto que contiene la información en lugar de extraer su información para procesarla fuera.

Por ejemplo, si tenemos que en un sistema de información de un hospital, cuando un episodio de ingreso de tipo CMA (Cirugía Mayor Ambulante) tiene una duración de mas de 12 horas, se considera como episodio de Hospitalización. Esto es un comportamiento intrínseco del objeto episodio, y debe implementarse como parte funcional de su propio comportamiento.

Ejemplo de programación Ask, donde la lógica de este comportamiento se realiza fuera:

```java
public class Episodio {
    private int numero;
    private String tipo;
    private Date fechaInicio;
    private Date fechaFinal;

    public Episodio(int numero, String tipo, Date fechaInicio){
        this.numero = numero;
        this.tipo = tipo;
        this.fechaInicio = fechaInicio;
    }

    public int getNumero() {
        return numero;
    }

    public void setNumero(int numero) {
        this.numero = numero;
    }

    public String getTipo() {
        return tipo;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

    public Date getFechaInicio() {
        return fechaInicio;
    }

    public void setFechaInicio(Date fechaInicio) {
        this.fechaInicio = fechaInicio;
    }

    public Date getFechaFinal() {
        return fechaFinal;
    }

    public void setFechaFinal(Date fechaFinal) {
        this.fechaFinal = fechaFinal;
    }

}

// Dentro del programa en el que utilizaríamos un objeto Episodio hariamos algo así al esti
// de programación preguntando, en lugar de pidiendo, implementando la lógica que queremos s
// el tipo de episodio en base a la duración del mismo

        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd H:m:s");
        Date fechaInicial=dateFormat.parse("2019-02-10 10:00:00");
        Episodio e = new Episodio(12345,"CMA",fechaInicial);
        Date fechaFinal=dateFormat.parse("2019-02-11 11:00:00");
        e.setFechaFinal(fechaFinal);
        int horas=(int) Math.floor(((e.getFechaFinal().getTime()-e.getFechaInicio().getTime(
        if(horas > 12) e.setTipo("HOS");

```

Ejemplo de programación Tell, donde este comportamiento estaría definido por el propio objeto:

```java
public class Episodio {
    private int numero;
    private String tipo;
    private Date fechaInicio;
    private Date fechaFinal;

    public Episodio(int numero, String tipo, Date fechaInicio){
        this.numero = numero;
        this.tipo = tipo;
        this.fechaInicio = fechaInicio;
    }

    public int getNumero() {
        return numero;
    }

    public void setNumero(int numero) {
        this.numero = numero;
    }

    public String getTipo() {
        return tipo;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

    public Date getFechaInicio() {
        return fechaInicio;
    }

    public void setFechaInicio(Date fechaInicio) {
        this.fechaInicio = fechaInicio;
    }

    public Date getFechaFinal() {
        return fechaFinal;
    }

    public void setFechaFinal(Date fechaFinal) {
        this.fechaFinal = fechaFinal;
        int horas=(int) Math.floor(((this.fechaFinal.getTime()- this.fechaInicio.getTime()),
        if(horas > 12) this.setTipo("HOS");

    }

}

// Y donde usariamos la clase Episodio delegaríamos la lógica del comportamiento
// del episodio al objeto episodio.

       SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd H:m:s");
        Date fechaInicial=dateFormat.parse("2019-02-10 10:00:00");
        Episodio e = new Episodio(12345,"CMA",fechaInicial);
        Date fechaFinal=dateFormat.parse("2019-02-11 11:00:00");
        e.setFechaFinal(fechaFinal);
// Y nada más.
```

Este ejemplo es muy sencillo que creo que sirve ilustrar el sentido del principio.

> En resumen, el principio «Tell, Don’t Ask» nos dice que, en lugar de pedir datos a los objetos, debemos decirles qué deben hacer y luego esperar el resultado de la operación. Viene a reforzar el concepto de encapsulación en la Orientación a Objetos.

Si en lugar de un tipo básico estamos acoplando con otra clase, es una buena práctica que las cosas que están estrechamente acopladas estén en el mismo componente. Limita el acoplamiento entre componentes.

Por otro lado, a veces se fuerza demasiado este principio e intentamos eliminar los getters/setters de nuestra clase. Hay que buscar un equilibrio, en muchas ocasiones es necesario que los objetos proporcionen información. Pero siempre hay que mantener la coherencia de la orientación a objetos, dejando la responsabilidad de operar con los atributos de un objeto al propio objeto.

algoritmos buenas prácticas

AUTOR

Jose Ramon Pascual

Ingeniero Informático de Sistemas por la Universidad de Málaga. Grado en Ingeniería Informática por la U. Internacional de La Rioja. Máster en Dirección de Proyectos, y Posgrados en Biotecnología, Bioingeniería, e-Health y Telemedicina por UNED. Comencé trabajando en automatización industrial en el sector de automoción, pero pronto pasé al sector sanidad, donde he trabajado durante 13 años en proyectos de sistemas de información vinculados a 79 hospitales de España de los servicios de sanidad de 15 comunidades autónomas, y Portugal. Tras 10 años de experiencia en interoperabilidad en sanidad con HL7 y **Mirth Connect**, he decidido divulgar experiencias sobre este motor.

‹ ¿PROVOCA AUTISMO LA VACUNA TRIPLE VÍRI... MLLP MINIMUM LOWER LAYER PROTOCOL ›

Etiquetas

Agujero negro (1) algoritmos (3) Arquitectura (3) Basura espacial (1) buenas prácticas (3) Cambio horario (2) carcinógenos (1) Como (8) Database (4) desinfección (1) e-Salud (7) ecologismo (5) epidemia (1) ER7 (2) Error (1) Event horizont telescope (1) eventos (5) excel (1) HL7 (7) Horizonte de sucesos (1) Howto (8) Interoperabilidad (17) IOT (1) JDBC (1) Layer (1) m87 (1) Mirth Connect (10) modularización (1) Open Source (5) optimización (2) Oracle (4) PL/SQL (2) productos químicos (1) Raspberry (1) Resiliencia (1) riesgos (1) Salud (6) SOA (2) SQL (2) tecnologica (1) toxicidad (1) Transcripción (1) vacunas (1) Web Service (2) XML (2)