Eva Otero
Yago Otero

Procura exemplos de código dos anteriores malos cheiros. Individualmente, en parellas ou tríos.
Cada exemplo debe conter:

1. Descripción do enunciado ou do que fai o código con malos cheiros.
2. Código fonte.
3. Identificación do mal cheiro que padece.
4. Descripción de como se debería solucionar.

A continuación enuméranse algúns malos cheiros que se poden atopar no código:

1. **Código duplicado (*duplicated code*):**

En este caso se repite la línea de impresión por lo que deberíamos crear un método donde podamos imprimir el resultado sin influir en cada método personalmente.

```
public class Calculadora {
    public int suma(int a, int b) {
        System.out.println("Realizando suma...");
        return a + b;
    }

    public int resta(int a, int b) {
        System.out.println("Realizando resta...");  // Línea duplicada
        return a - b;
    }

    public int multiplicacion(int a, int b) {
        System.out.println("Realizando multiplicación...");  // Línea duplicada
        return a * b;
    }
}
```

2. **Complexidade artificial (*contrived complexity*):** consiste en utilizar patróns de deseño complexos cando se podería utilizar un deseño máis sinxelo ou menos complexo.

```
public class OverlyEngineeredHelloWorld {
    public static void main(String[] args) {
        Message message = new HelloWorldMessage();
        System.out.println(message.getContent());
    }
}

interface Message {
    String getContent();
}

class HelloWorldMessage implements Message {
    @Override
    public String getContent() {
        return "Hello World!";
    }
}
```

A nivel de método:

Método longo (*long method*):

En este código podemos apreciar como hemos juntado todo el método en uno mismo en vez de seguir la regla de “Divide y vencerás” que comúnmente podremos utilizar en la programación

```
public class Factura {
    public void imprimirFactura(Cliente cliente,
List<Producto> productos) {
        validarDatos(cliente, productos);
        double[] totales = calcularTotales(productos);
        imprimirCabecera(cliente);
        imprimirProductos(productos);
        imprimirTotales(totales[0], totales[1], totales[2]);
    }

    private void validarDatos(Cliente cliente, List<Producto>
productos) {
        if (cliente == null || productos == null ||
productos.isEmpty()) {
            throw new IllegalArgumentException("Datos
inválidos");
        }
    }

    private double[] calcularTotales(List<Producto>
productos) {
        double subtotal = 0;
        double impuestos = 0;
        for (Producto p : productos) {
            subtotal += p.getPrecio();
            impuestos += p.getPrecio() * 0.16;
        }
        return new double[]{subtotal, impuestos, subtotal +
impuestos};
    }

    private void imprimirCabecera(Cliente cliente) {
        System.out.println("=== FACTURA ===");
        System.out.println("Cliente: " +
cliente.getNombre());
        System.out.println("RFC: " + cliente.getRfc());
        System.out.println("Fecha: " + LocalDate.now());
    }

    // ... otros métodos extraídos ...
```

```
}
```

Cadeas de mensaxes (*message chains*): un método chama a outro método, que chama a outro método e este método chama a outro, etc.

```
public class Conversacion {

    public void iniciarConversacion() {
        String nombre = obtenerNombre();
        saludar(nombre);
    }

    private String obtenerNombre() {
        System.out.print("Por favor, ingresa tu nombre: ");
        Scanner scanner = new Scanner(System.in);
        return scanner.nextLine();
    }

    private void saludar(String nombre) {
        System.out.println("¡Hola, " + nombre + "!");
        preguntarEstado();
    }

    private void preguntarEstado() {
        System.out.println("¿Cómo te encuentras hoy?");
        responder();
    }

    private void responder() {
        System.out.println("(Simulando respuesta del usuario)");
        System.out.println("¡Que tengas un buen día!");
    }

    public static void main(String[] args) {
        Conversacion chat = new Conversacion();
        chat.iniciarConversacion();
    }
}
```

Demasiados parámetros (*too many parameters*):
En el siguiente ejemplo podemos observar como tenemos nuestra clase Cliente donde para crear cada cliente debemos añadir demasiados parámetros que dificultan la lectura del código y puede llevar a dar muchos errores por confusión o similar

```
public class ClienteService {

    // ¡Método con 8 parámetros! Difícil de usar y mantener.
    public void crearCliente(String nombre, String apellido,
String email,
                             String telefono, String direccion,
String ciudad,
                             String codigoPostal, String pais) {
        // Validaciones...
        Cliente cliente = new Cliente(nombre, apellido, email,
telefono,
                                      direccion, ciudad,
codigoPostal, pais);
        database.save(cliente);
    }
}
```

3. **Liña de código excesivamente longa (*God line*):**
   Cuando contamos con varias condiciones es mejor ir yendo dividiéndolas poco a poco antes de crear una condición extremadamente larga.

```
if (usuario != null && usuario.isActive() &&
usuario.hasPermission(Permission.ADMIN) {
adminDashboard.mostrar(); }
```

4. **Excesiva devolución (*excessive returner*):** consiste nun método que devolve máis datos dos necesarios.

```
public class ServicioCliente {

    // Método que devuelve muchos datos cuando solo se necesita
uno
    public String[] getInfoCliente(int idCliente) {
        String[] datos = new String[5];
        datos[0] = "Juan Pérez";       // Nombre
        datos[1] = "juan@email.com";    // Email
        datos[2] = "Activo";            // Estado
        datos[3] = "Premium";           // Tipo de cliente
        datos[4] = "555-1234";         // Teléfono
```

```
        return datos; // Devuelve 5 datos cuando normalmente solo
se usa 1
    }

    public static void main(String[] args) {
        ServicioCliente servicio = new ServicioCliente();
        String[] info = servicio.getInfoCliente(101);

        // Normalmente solo usamos el email
        System.out.println("Email del cliente: " + info[1]);
    }
}
```

5. **Tamaño do identificador (*identifier size*):**
   Cuando tenemos métodos con nombres excesivamente largos dificulta a la hora de llamarlos o similar , si por ejemplo tenemos un método que nos devuelva un verdadero o falso pero en cambio es excesivamente largo de nada sirve

```
String nombreDelClienteQueRealizaLaCompraEnLaTiendaOnline; // 52
caracteres
int numeroTotalDeProductosEnElCarritoDeComprasDelUsuario; // 49
caracteres
```

A nivel de clase:

6. **Clase grande (*large class*):** consiste en ter unha clase con demasiados atributos, métodos ou instancias. Isto débese a que a clase ten asignadas máis tarefas das que deberían ser asignadas.
7. **Clase demasiado simple (*freeloader*):** consiste en ter unha clase con moi poucas responsabilidades. Moitas veces son clases que só teñen atributos e métodos de acceso (set) e consulta (get).
8. **Envexa de funcionalidade (*feature envy*):** hai un método nunha clase que parece máis interesado noutra clase que na clase na que está, é dicir, un método que utiliza en exceso os métodos doutra clase. Adoita resolverse movendo o método á clase cuxos elementos utiliza máis.
9. **Cirurxía a tiros (*shotgun surgery*):** un cambio nunha clase leva a cambios en moitas outras clases.
10. **Código diverxente (*divergent code*):** ao facer un cambio no sistema, unha clase sofre demasiados tipos de cambios.
11. **Grupo de datos (*data clump*):** ocorre cando os conxuntos de datos se agrupan en varias partes do programa. Probablemente sexa máis apropiado agrupar as variables nun só obxecto.
12. **Intimidade inadecuada (*inappropiate intimacy*):** ocorre cando unha clase depende dos detalles de implementación doutra clase.

13. **Legado rexeitado (*refused bequest*):** unha clase non usa métodos e atributos da superclase, o que normalmente indica que a xerarquía de clases se creou incorrectamente.
14. **Complexidade ciclomática (*cyclomatic complexity*):** esta é unha clase con demasiadas ramas e bucles. Isto significa que o método ou métodos afectados por este problema deberían dividirse en varios métodos ou simplificarse.

Procura exemplos de código dos anteriores malos cheiros.