Primer parcial refactoring

Ejercicio 2 - Refactoring

OO2 -1er recuperatorio- 29/06/2024

Para el siguiente código, realice las siguientes tareas:

(i) indique que mal olor presenta

(ii) indique el refactoring que lo corrige

(iii) aplique el refactoring (modifique el código)

Si vuelve a encontrar un mal olor, retorne al paso (i).

Nota: Haga los cambios que considere necesarios.

```
1. public class Pago {
3.    private List<Producto> productos;
4.    private String tipo;
5.    private static final double ADICIONAL_TARJETA = 1000.0;
6.    private static final double DESCUENTO_EFECTIVO = 2000.0;

7.    public Pago(String tipo, List<Producto> productos) {
8.       this.productos = productos;
9.       this.tipo = tipo;
10.   }

12.   public double calcularMontoFinal() {
13.      double total = 0.0;
14.      if (this.tipo == "EFECTIVO"){
15.         for (Producto producto: this.productos){
16.            total = total + producto.getPrecio() + (producto.getPrecio() * producto.getIVA());
17.         }
18.      if (total > 100000){
19.            total = total - DESCUENTO_EFECTIVO;
20.         }
21.      }
22.      else if (this.tipo == "TARJETA"){
23.         for (Producto producto: this.productos){
24.            total = total + producto.getPrecio() + (producto.getPrecio() * producto.getIVA())
25.         }
26.         total = total + ADICIONAL_TARJETA;
27.      }
28.      return total;
29.   }
30. }
```

```
1.  public class Producto {
2.     private double precio;
3.     private double IVA;
4.
5.     public Producto(double precio, double IVA) {
6.        this.precio = precio;
7.        this.IVA = IVA;
8.     }
9.
10.    public double getPrecio() {
11.       return this.precio;
12.    }
```

```
13.   public double getIVA() {
14.         return this.IVA;
15.   }
16. }
```

Code smell encontrados:

- Switch statement
- Reinventar la rueda
- Feature Envy
- Long Method

Aplicando refactoring

Para el caso de reinventar la rueda, este se encuentra en la linea 15 a 17, y 23 a 25.

Refactoring a aplicar: replace loop with pipeline

Pasos: Primero creo el mismo algoritmo pero en vez de utilizar un for utilizo streams.
muevo el algoritmo creado y lo posiciono en el mismo lugar en el que se encuentra el loop,
elimino el loop.

Resultado:

```
total = productos.stream()
            .mapToDouble(producto → producto.getPrecio() +
               producto.getPrecio() * producto.getIva())
            .sum();
```

Refactoring a aplicar para el feature envy: Move Method

Pasos: Creo un nuevo método llamado calcularPrecio() en la misma clase (clase Pago) el cual va a recibir como parámetro un objeto de tipo Producto.
Muevo el calculo que se encuentra dentro del mapToDouble adentro del método.
Muevo el método a la clase pago, elimino el parámetro pasado y los accesos, ahora los accesos van a cambiar y se van a utilizar las variables dentro del mismo objeto.

cambio la llamada que hace de producto en .mapToDouble y la cambio por la llamada al producto.calcularPrecio(). Elimino en la clase Producto los metodos getIva y getPrecio, ya son redundantes porque no se van a utilizar

Resultado:

```
// Dentro de la clase pago
public double calcularTotal() {
    return this.precio + (this.precio * this.IVA);
}

// pipeline correcido
// .mapToDouble(Producto::calcularTotal).sum();
```

Nuevo refactoring a aplicar: Replace conditional with strategy method

Paso 1: creo la jerarquia de clases necesaria para el problema: Creo una nueva interfaz llamada IMetodoPago el cual va a tener un metodo llamado aplicarDescuento(double total).
Creo las subclases necesarias (Efectivo, Tarjeta), Hago extract method llamado calcularDescuento(double total) y muevo el if de total que se encuentra dentro del if de efectivo. Una vez hecho eso muevo el metodo a la subclase efectivo.
Procedo a hacer lo mismo pero en este caso con el tipo Tarjeta.

Una vez implementado los metodos correctamente creo una nueva variable de instancia de tipo IMetodoPago metodoPago, creo un metodo llamado setMetodoPago el cual el usuario puede cambiar el metodoDePago en tiempo de ejecucion. Elimino codigo repetivo y simplemente hago una vez el pipeline del calculo total de productos. Para posterior a eso retornar el resultado de la llamada al metodo aplicarDescuento, pasandole como parámetro total.
Y por ultimo, paso como parámetro en el método constructor IMetodo metodoPago

Elimino las constantes ADICIONAL_TARJETA Y DESCUENTO_EFECTIVO.
Elimino la variable tipo y todas sus llamadas dentro del código.

Código finalizado

```
public class Pago {
    private List<Producto> productos;
    private IMetodoPago metodoPago;

    public Pago(IMetodoPago metodoPago, List<Producto> productos) {
        this.productos = productos;
        this.metodoPago = metodoPago;
    }

    public void setMetodoPago(IMetodoPago metodoPago) {
        this.metodoPago = metodoPago;
    }

    public double calcularMontoFinal() {
        double total = 0.0;
        total = productos.stream()
                .mapToDouble(Producto::calcularTotal)
                .sum();
        return metodoPago.aplicarDescuento(total);
    }

}

public class Producto {
    private double precio;
    private double IVA;

    public Producto(double precio, double IVA) {
        this.precio = precio;
        this.IVA = IVA;
    }

    public double calcularTotal() {
        return this.precio + (this.precio * this.IVA);
```

```
    }
}


public interface IMetodoPago {
    public double aplicarDescuento(double total);
}


public class Efectivo extends IMetodoPago {

    public double aplicarDescuento(double total) {
        if (total > 100000) {
            return total - 2000;
        }
        return total;
    }

}

public class Tarjeta extends IMetodoPago {
    public double aplicarDescuento(double total) {
        return total + 1000
    }
}
```