Parcial primer recuperatorio 29-06-2024

Para el siguiente codigo realice las siguientes tareas:
1: Indique que mal olor presenta
2: Indique el refactoring que lo corrige
3: Aplique el refactoring (modifique el codigo)
Si vuelve a encontrar un mal olor, retome al paso 1

Refactorings:

(1) Refactoring
Bad smell: Duplicated code
Refactoring: Extract method
En las lineas 15 hasta 17 y 23 hasta 25 se esta repitiendo la misma logica para calcular el precio de los productos. Se deberia extraer esa logica en un metodo y llamar al metodo dos veces para evitar duplicar codigo. Luego debemos

```
public class Pago
{
      ...codigo...
    if (this.tipo =='EFECTIVO') {
    total = this.calcularPrecioProductos();
        if(total > 1000000)
        total = total - DESCUENTO_EFECTIVO;
    }
    else if(this.tipo =='TARJETA'){
    total = this.calcularPrecioProductos();
    total = total + ADICIONAL_TARJETA;
    }
      ...codigo...


    private double calcularPrecioProductos()
     {
        double total;
        for (Producto producto : this.productos)
        {
            total = total + producto.getPrecio() + (producto.getPrecio() *
producto.getIva());
        }
        return total;
     }
```

```
}
```

(2) refactoring
Bad smell: Feature Envy / envidia de atributos
Refactoring que lo corrige: Extract Method
En el metodo calcularPrecioProductos se esta accediendo al precio y al iva de la clase Producto para calcular el precio, es envidia de atributos. El responsable de calcular el precio DEBE ser la clase Producto.

- Creamos el metodo getPrecioFinal en la clase producto.

```
public class Producto
{
   private double precio;
   private double IVA;
   ... codigo ...
   public double getPrecioFinal()
   {
       return this.precio + (this.precio * this.Iva );
   }
   ... codigo ...
}
```

- En el metodo calcularPrecioProductos, le decimos al producto que calcule su precio y nos lo de.

```
private double calcularPrecioProductos()
{
   double total;
   for (Producto producto : this.productos)
   {
       total += producto.getPrecioFinal();
   }
   return total;
}
```

(3) Refactoring
Bad smell: Reinventar la rueda
Refactoring: Use Stream Pipelines
En el metodo calcularPrecioProductos de la clase Pago, tenemos un For, el cual lo vamos a reemplazar por funciones que nos proporciona Java, como es el caso de los Streams

```
 private double calcularPrecioProductos()
 {
    return this.productos.stream()
            .mapToDouble(producto -> producto.getPrecioFinal())
            .sum();
 }
```

(4) Refactoring:
Bad Smell: Switch Statement
Refactoring: Replace conditional with Polymorphism
En las lineas 14 y 22, se esta preguntando por "tipos" de clases. La solucion es crear una jeraquia de tipos de "MetodosDePago". Se debe aplicar extract class, creando las clases "Tarjeta" y "Efectivo", y hacer que ambas clases implementen la interfaz "MetodoDePago" que contiene un metodo con nombre "calcularPrecioFinal".
En conjunto con este refactoring, aplicaria move field, es decir los campos estaticos de la clase Pago, los muevo hacia la clase que correspondan. Por ejemplo el adicional tarjeta lo muevo a la clase tarjeta.

```
public interface MetodoDePago
{
    public double calcularPrecioFinal(double precioDeProductos);
}

public class Tarjeta implements MetodoDePago
{
    private static final double = ADICIONAL_TARJETA = 1000.0;

    public double calcularPrecioFinal(double precioDeProductos)
    {
        return precioDeProductos + ADICIONAL_TARJETA;
    }
}

public class Efectivo implements MetodoDePago
{
    private static final double DESCUENTO_EFECTIVO= 2000.0
    public double calcularPrecioFinal(double precioDeProductos)
    {
        if(precioDeProductos > 100000) precioDeProductos -=
DESCUENTO_EFECTIVO;
        return precioDeProductos;
    }
}
```

```
public class Pago
{
    private List<Producto> productos;
    private MetodoDePago metodoDePago;
    public Pago (List<Producto> productos, MetodoDePago metodoDePago)
    {
        this.productos = productos;
        this.metodoDePago = metodoDePago;
    }

    public double calcularMontoFinal()
    {
        double total = 0.0;
        total =
this.metodoDePago.calcularMontoFinal(this.calcularPrecioProductos);
        return total;
    }

    private double calcularPrecioProductos()
     {
        return this.productos.stream()
                .mapToDouble(producto -> producto.getPrecioFinal())
                .sum();
     }
}
```

(5) Refactoring:
Bad Smell: Temporary field
Refactoring: Replace temporary field with query
En el metodo calcularMontoFinal de la clase Pago, elimino la variable total, y hago que el metodo retorne directamente el valor que retorna metodoDePago

```
    public double calcularMontoFinal()
    {
        return
this.metodoDePago.calcularMontoFinal(this.calcularPrecioProductos);
    }
```

Refactorings terminados.

Codigo completo:

```
public interface MetodoDePago
{
```

```
    public double calcularPrecioFinal(double precioDeProductos);
}

public class Tarjeta implements MetodoDePago
{
    private static final double = ADICIONAL_TARJETA = 1000.0;
    public double calcularPrecioFinal(double precioDeProductos)
    {
        return precioDeProductos + ADICIONAL_TARJETA;
    }
}

public class Efectivo implements MetodoDePago
{
    private static final double DESCUENTO_EFECTIVO= 2000.0
    public double calcularPrecioFinal(double precioDeProductos)
    {
        if(precioDeProductos > 100000) precioDeProductos -=
DESCUENTO_EFECTIVO;
        return precioDeProductos;
    }
}

public class Pago
{
    private List<Producto> productos;
    private MetodoDePago metodoDePago;
    public Pago (List<Producto> productos, MetodoDePago metodoDePago)
    {
        this.productos = productos;
        this.metodoDePago = metodoDePago;
    }

    public double calcularMontoFinal()
    {
        return
this.metodoDePago.calcularMontoFinal(this.calcularPrecioProductos);
    }

    private double calcularPrecioProductos()
     {
        return this.productos.stream()
                .mapToDouble(producto -> producto.getPrecioFinal())
                .sum();
     }
}
```

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

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

    public double getPrecioFinal()
    {
        return this.precio + (this.precio * this.IVA);
    }


    public double getPrecio()
    {
        return this.precio;
    }

    public double getIVA()
    {
        return this.IVA;
    }

}
```