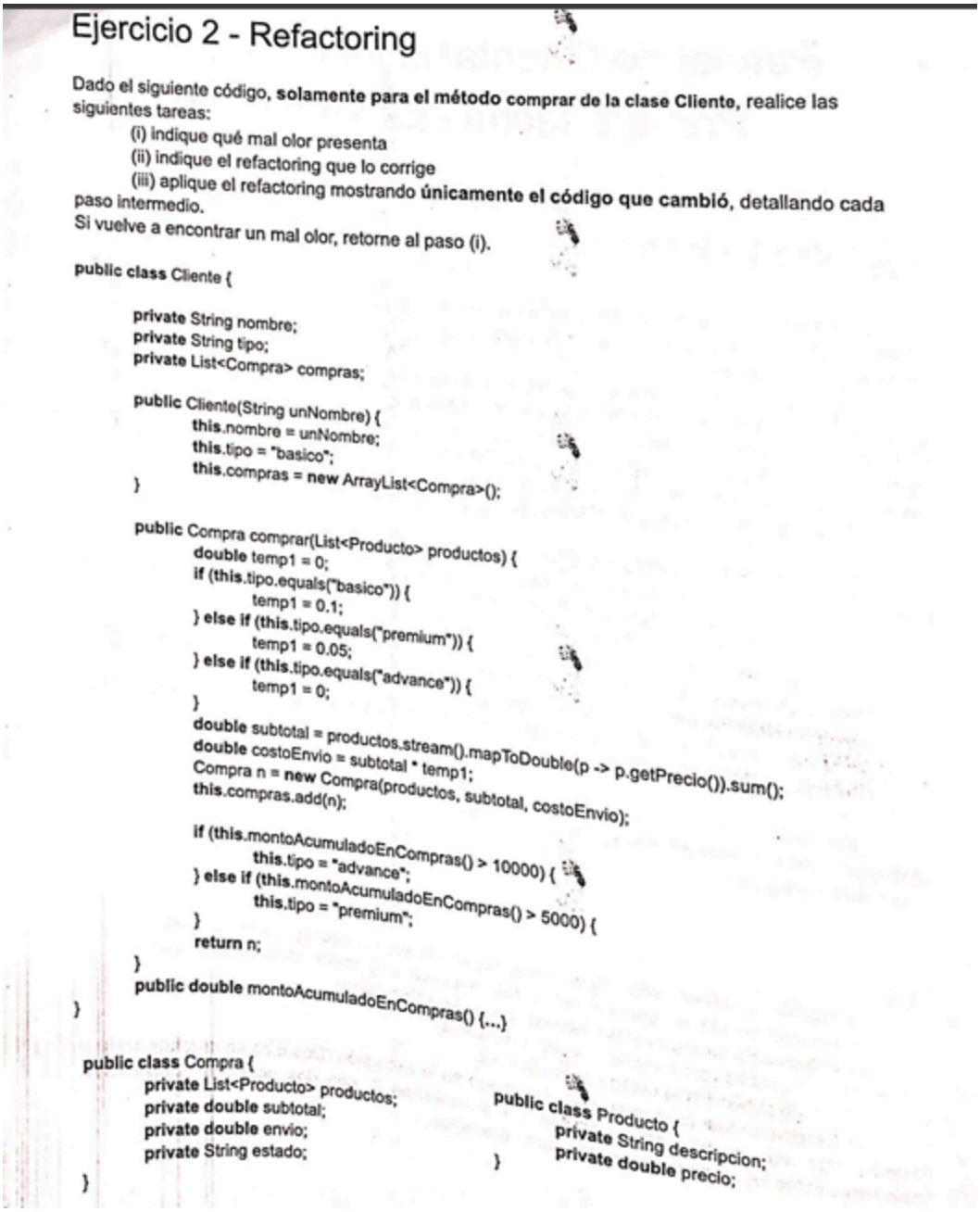
Refactoring



Ejercicio 2 - Refactoring

Dado el siguiente código, **solamente para el método comprar de la clase Cliente**, realice las siguientes tareas:

(i) indique qué mal olor presenta
(ii) indique el refactoring que lo corrige
(iii) aplique el refactoring mostrando **únicamente el código que cambió**, detallando cada paso intermedio.
Si vuelve a encontrar un mal olor, retorne al paso (i).

```
public class Cliente {

    private String nombre;
    private String tipo;
    private List<Compra> compras;

    public Cliente(String unNombre) {
        this.nombre = unNombre;
        this.tipo = "basico";
        this.compras = new ArrayList<Compra>();
    }

    public Compra comprar(List<Producto> productos) {
        double temp1 = 0;
        if (this.tipo.equals("basico")) {
            temp1 = 0.1;
        } else if (this.tipo.equals("premium")) {
            temp1 = 0.05;
        } else if (this.tipo.equals("advance")) {
            temp1 = 0;
        }
        double subtotal = productos.stream().mapToDouble(p -> p.getPrecio()).sum();
        double costoEnvio = subtotal * temp1;
        Compra n = new Compra(productos, subtotal, costoEnvio);
        this.compras.add(n);

        if (this.montoAcumuladoEnCompras() > 10000) {
            this.tipo = "advance";
        } else if (this.montoAcumuladoEnCompras() > 5000) {
            this.tipo = "premium";
        }
        return n;
    }
    public double montoAcumuladoEnCompras() {...}
}

public class Compra {
    private List<Producto> productos;
    private double subtotal;
    private double envio;
    private String estado;
}

public class Producto {
    private String descripcion;
    private double precio;
}
```

Lo voy a resolver como se resuelve en este año (2025). Primero marcamos todos los code smells, y despues empezamos a refactorizar.

Code Smells:

(1): Switch Statement :
Se pregunta el tipo de cliente para calcular el porcentaje

(2): Feature Envy :
La clase Cliente esta envidiando los atributos para calcular el subtotal y el costo de envio. Deberia delegar esta accion a la clase Compra.

(3) Long Method:
Una solucion es aplicar Extract Method, y ademas se debe aplicar polimorfismo para resolver los problemas de tipos.

(4) Nombre poco autoexplicativo:
La variable temp1 no indica bien cual es su objetivo.

(5) Nombre poco autoexplicativo:
La variable temporal 'n' dentro del metodo Comprar de la clase Cliente, no indica bien cual es su objetivo.

Refactorings:

(1) Code Smell: Switch Statement
Refactoring: Aplicar Replace Conditional With Polymorphism

Creamos las siguientes clases e interfaces:

- Creamos una interfaz 'tipoCliente'
- Creamos una clase concreta 'clienteBasico'
- Creamos una clase concreta 'clientePremium'
- Creamos una clase concreta 'clienteAdvance'

Por cada rama del condicional hacemos:

- Extract Method
- Creamos el metodo public double calcularPorcentaje( )
- Aplicamos Move Method para moverlo a la rama correspondiente

  Este paso lo aplicamos para las ramas de clienteBasico, clientePremium y clienteAdvance
- Como las tres clases comparten la misma firma del metodo y cada una la implementa de una forma distinta. Creamos una firma de metodo en la interfaz 'tipoCliente'.
- Hacemos que las clases de la rama del condicional, implementen la interfaz 'tipoCliente'
- Modificamos de la clase Cliente:
  - La variable de instancia String tipo, ahora es private TipoCliente tipoCliente
  - Modificamos el tipo de cliente en el constructor, lo inicializamos con un new ClienteBasico
  - Eliminamos las ramas condicionales del primer bloque de if, reemplazandola por polimorfismo.

        - ahora hacemos double temp1 = tipoCliente.calcularPorcentaje();

(2) Code Smell: Feature Envy (Calculo del subtotal)

Refactoring a aplicar: Extract Method y Move Method

- Aplicamos Extract Method en la linea donde se calcula el subtotal
    - En la clase Cliente, creamos el metodo con firma public double calcularSubtotal()
    - Movemos la logica del calculo dentro del metodo
    - Aplicamos Move Method
        - Lo movemos a la clase Compra

(2) Code Smell: Feature Envy (Calculo del costoEnvio)
Refactoring aplicar Extract Method y Move Method

- Aplicamos Extract Method en la linea donde se calcula el costo de envio
    - En la clase Cliente, creamos el metodo con firma public double calcularCostoEnvio()
    - Movemos la logica del calculo dentro del metodo
    - Aplicamos Move Method
        - Lo movemos a la clase Compra
        - Debemos pasarle por parametro 'temp1'

(3) Code Smell: Long Method
Refactoring Aplicar Extract Method

- Aplicamos Extract Method
    - Creamos un metodo private void chequearCambioTipoCliente(double montoAcumulado)
    - Movemos la logica del segundo bloque if a este methodo.
    - Modificamos la forma en que le da valor al tipo, ya no es mas un String, sino se utiliza la subclase de TipoCliente Correspondiente
    - Se reemplazan las lineas de logica mencionadas,. por una invocacion al metodo creado, pasandole por parametro el montoAcumulado.

(4) Code Smell: Nombre poco autoExplicativo
Refactoring Aplicar Rename Field

- Cambiamos el nombre de 'temp1' a 'factorEnvio'
- cambiamos las referencias de 'temp1' a 'factorEnvio'

(5) Code Smell: Nombre poco autoExplicativo
Refactoring Aplicar Rename Temporary Field

- Cambiamos el nombre de 'n' a 'compra'
- cambiamos las referencias de 'n' a 'compra'

```
public class Cliente {
    private String nombre;
    private TipoCliente tipoCliente;
    private List<Compra> compras;

    public Cliente (String unNombre)
    {
        this.compras = new LinkedList<>();
        this.nombre = unNombre;
        this.tipoCliente = new ClienteBasico();
    }

    public Compra comprar (List<Producto> productos)
    {
        double factorEnvio = tipoCliente.calcularPorcentaje();
        Compra compra = new Compra(productos,factorEnvio);
        this.compras.add(compra);
        this.chequearCambioTipoCliente(this.montoAcumuladoEnCompras());

        return compra;
    }

    public double montoAcumuladoEnCompras()
    {
        return compras.stream()
                .mapToDouble(compra -> compra.getSubtotal())
                .sum();
    }

    private void chequearCambioTipoCliente(double montoAcumuladoEnCompras)
    {
        if (montoAcumuladoEnCompras > 10000) this.tipoCliente = new
ClienteAdvance();
        else if (montoAcumuladoEnCompras > 5000) this.tipoCliente = new
ClientePremium();
    }
}

public interface TipoCliente {
    double calcularPorcentaje();
}
```

```java
public class ClienteAdvance implements TipoCliente {
    @Override
    public double calcularPorcentaje() {return 0;}
}

public class ClienteBasico implements  TipoCliente{
    @Override
    public double calcularPorcentaje()
    { return 0.1;  }
}

public class ClientePremium implements  TipoCliente{
    @Override
    public double calcularPorcentaje()
    { return 0.05;  }
}

public class Compra {
    private List<Producto> productos;
    private double subtotal;
    private double envio;
    private String estado;

    public Compra(List<Producto> productos, double factorEnvio)
    {
        this.productos = productos;
        this.subtotal = this.calcularSubtotal();
        this.envio = this.calcularCostoEnvio(factorEnvio);
        this.estado = "unEstado";
    }

    private double calcularSubtotal()
    {
        return productos.stream()
                .mapToDouble(prod -> prod.getPrecio())
                .sum();
    }

    private double calcularCostoEnvio(double factorEnvio)
    {
        return  this.subtotal * factorEnvio;
    }

    public double getSubtotal()
    {
```

```
        return this.subtotal;
    }
}

public class Producto {
    private String descripcion;
    private double precio;

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

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

}
```