ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERIA EN ELECTRICIDAD Y COMPUTACION

**Taller de Principios SOLID – Diseño de Software**

# Objetivos:

* Identificar violaciones a los principios SOLID.
* Corregir código que viole principios SOLID.

Una pequeña empresa de postres ha comenzado a armar el sistema de todo su negocio. Los primeros productos que han sacado a la venta son pasteles y helados.

Se sabe también que ya tienen como calcular los precios de cada producto que vende. Los productos tienen un costo parcial fijo, al cual se les aumenta el valor del IVA. A cada postre se le puede añadir distintos aderezos que tienen un costo de 0.50 ctvs. cada uno. Sumando este valor adicional, al anterior se obtiene el valor final de cada producto que venden.

Todos los postres se realizan con leche entera, sin embargo, el negocio ofrece la facilidad de cambiar el tipo de leche a descremada o deslactosada, según el cliente lo prefiera sin ningún costo adicional.

Luego de unos días, una persona que conoce sobre diseño de software ha revisado el sistema, y ha identificado que existen ciertos principios SOLID están siendo violados por el sistema desarrollado.

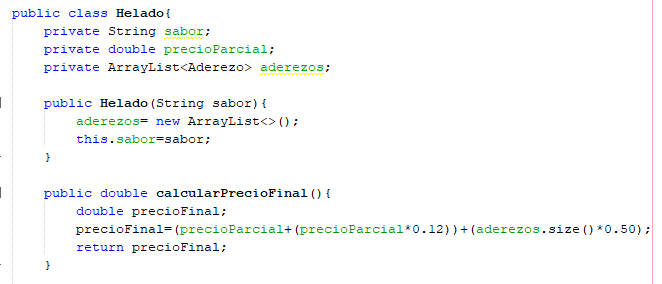
# Indicaciones

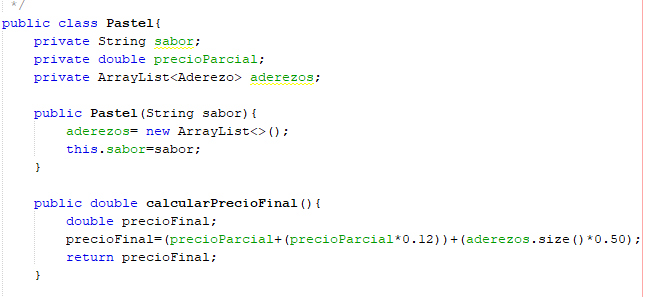
1. Identifique los principios SOLID que no están siendo respetados explicando el porqué.
2. Corrija el código para que solucionar los problemas previamente mencionados, creando un nuevo paquete donde si se respete los principios SOLID.
3. Por cada principio violado, muestre el código que lo incumple y luego muestre y explique como solucionó el problema con su código.

Puede crear clases o interfaces adicionales si considera necesario, pero no elimine el código para poder hacer la comparación final.

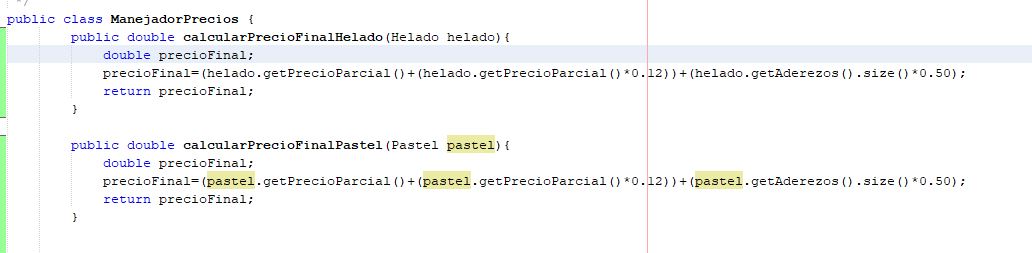
# Desarrollo

1. SRP: calcularPrecioFinal no es una responsabilidad de los postres



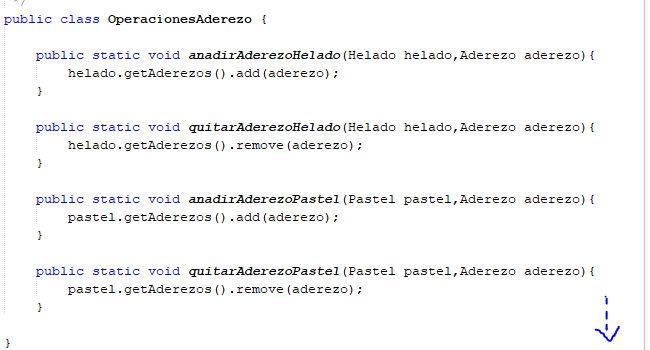


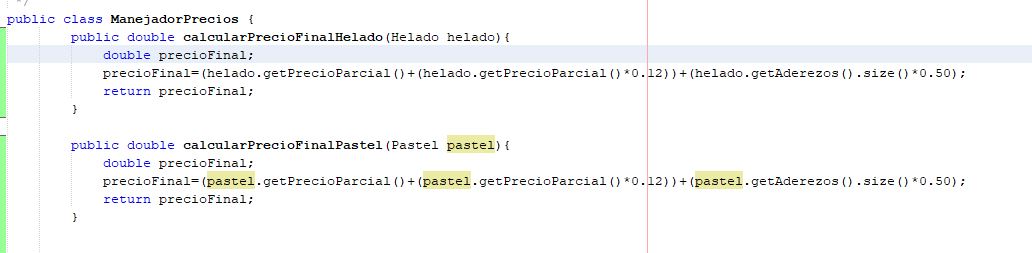
Solución: Crear una clase nueva que controle el cálculo del precio final (ManejadorPrecios)

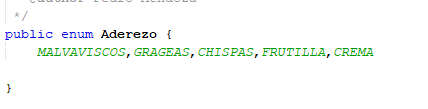
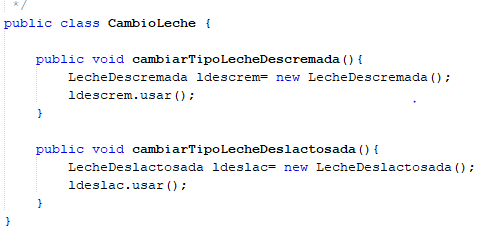


1. OCP: OperacionesAderezo tenía que ser modificado cada vez que se agregaba un postre nuevo.

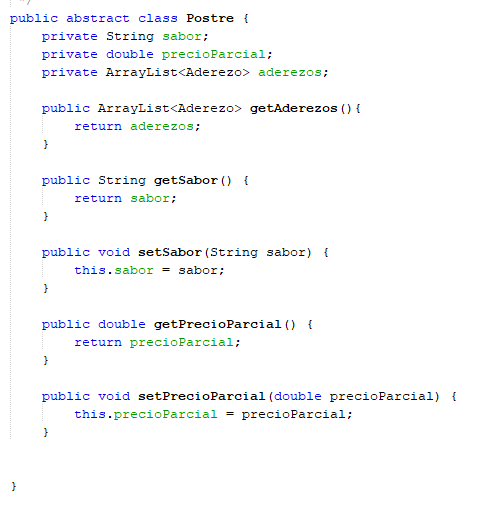
ManejadorPrecios también debía ser modificada modificado cada vez que se agregaba un postre nuevo. CambioLeche también necesitaba modificación cada vez que se agrega un nuevo tipo de leche. Aderezo era un enum, por lo cual al agregar un nuevo aderezo había que cambiar la clase Aderezo.

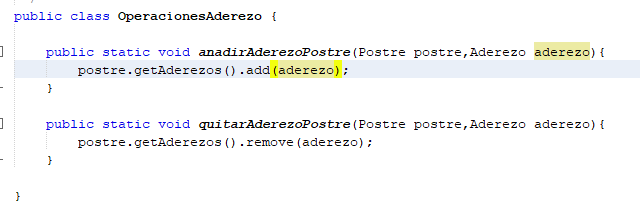


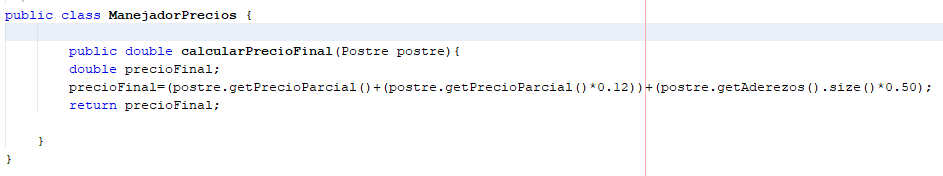




Solución: Crear una clase abstracta Postre y que cada nuevo postre que se agregue herede de esa clase. Utilizar esta clase en los métodos de OperacionesAderezo y ManejadorPrecios.

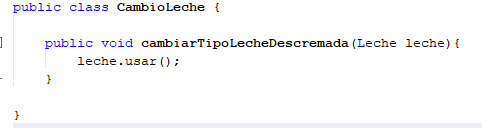






Agregar interfaz Leche y que las leches implementen esa interfaz. Modificar CambioLeche para que use una Leche en sus métodos.





Convertimos Aderezo en clase con un atributo nombre que le permita identificarse y usar esa clase al elaborar un postre, de manera que la clase Aderezo no se modifica, solo se instancia.

