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

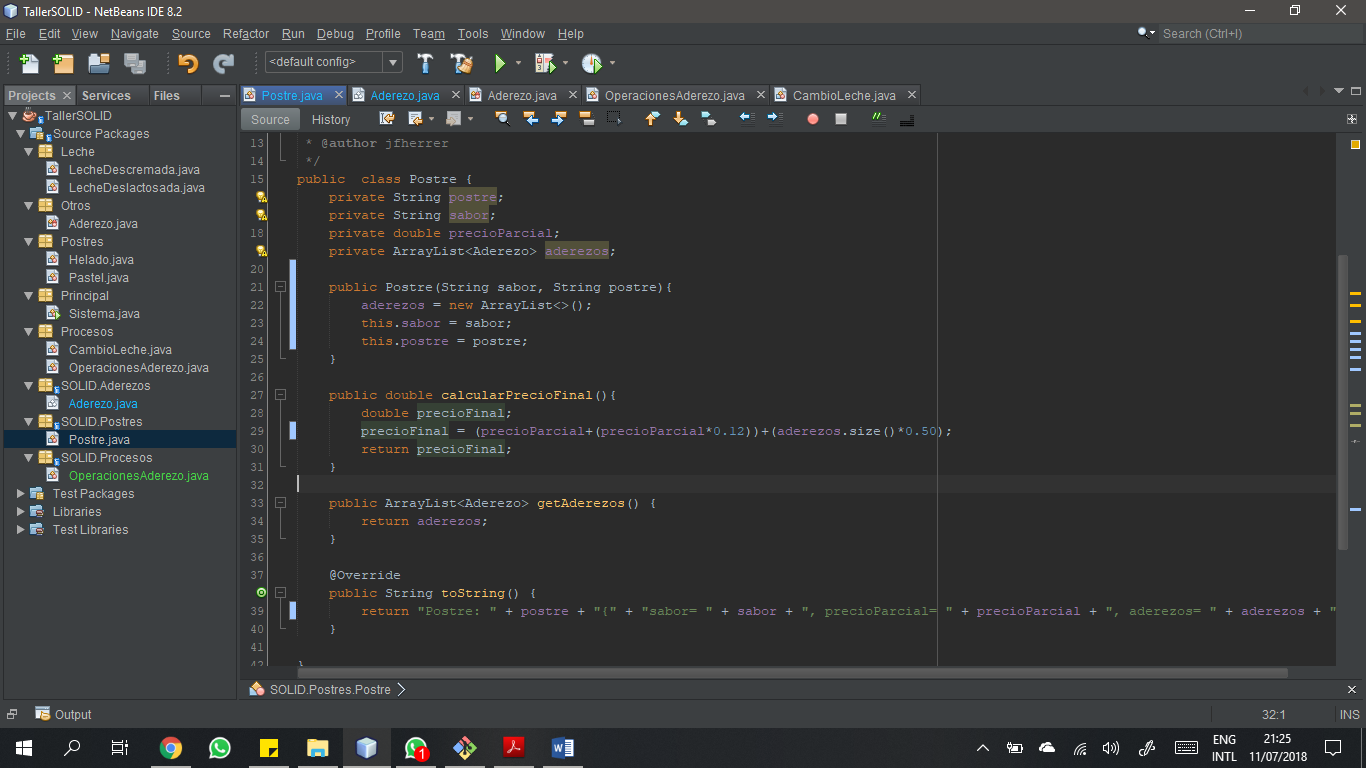
1. Identifique los principios SOLID que no están siendo respetados explicando el porqué.
2. Corrija el código para que solucionar los problemas previamente mencionados, creando un nuevo paquete donde si se respete los principios SOLID.
3. Por cada principio violado, muestre el código que lo incumple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

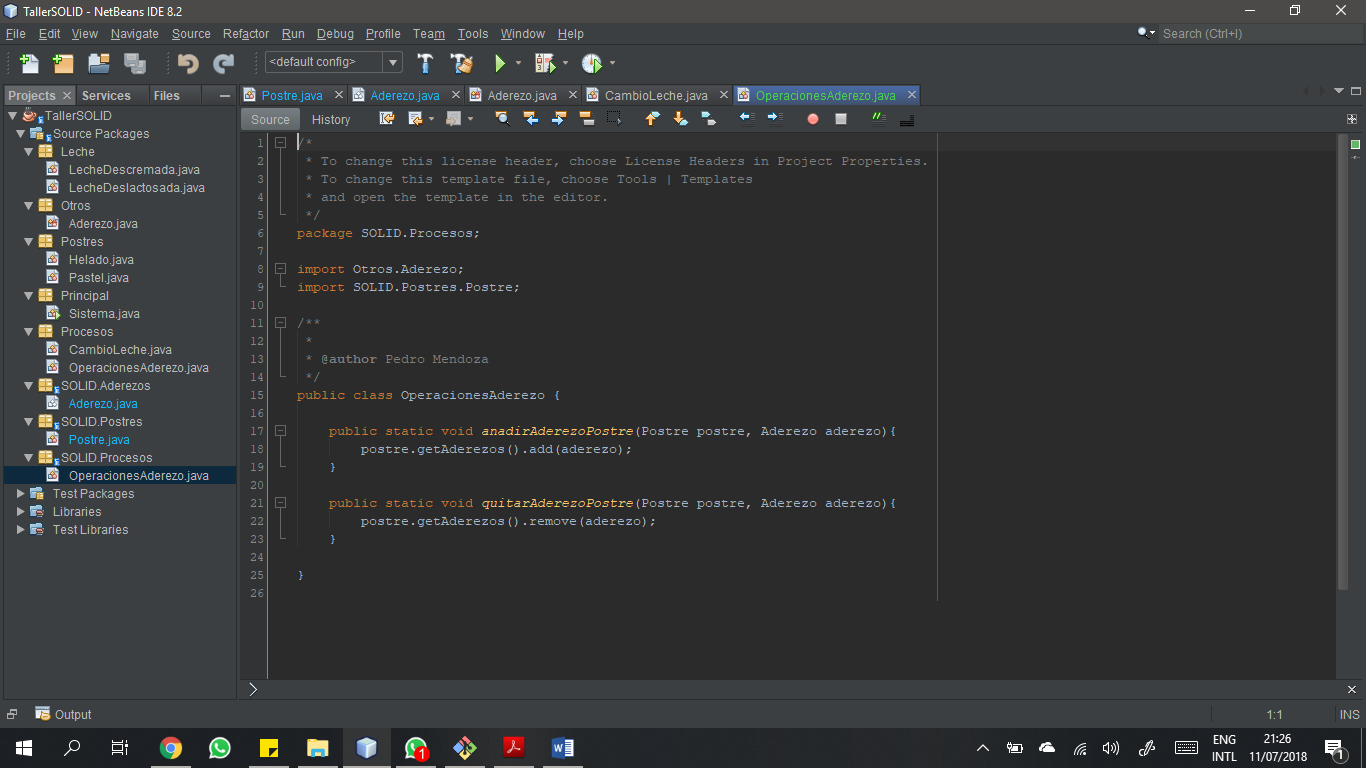
Puede crear clases o interfaces adicionales si considera necesario, pero no elimine el código para poder hacer la comparación final.

# Desarrollo

Aquí desarrolle lo anterior.

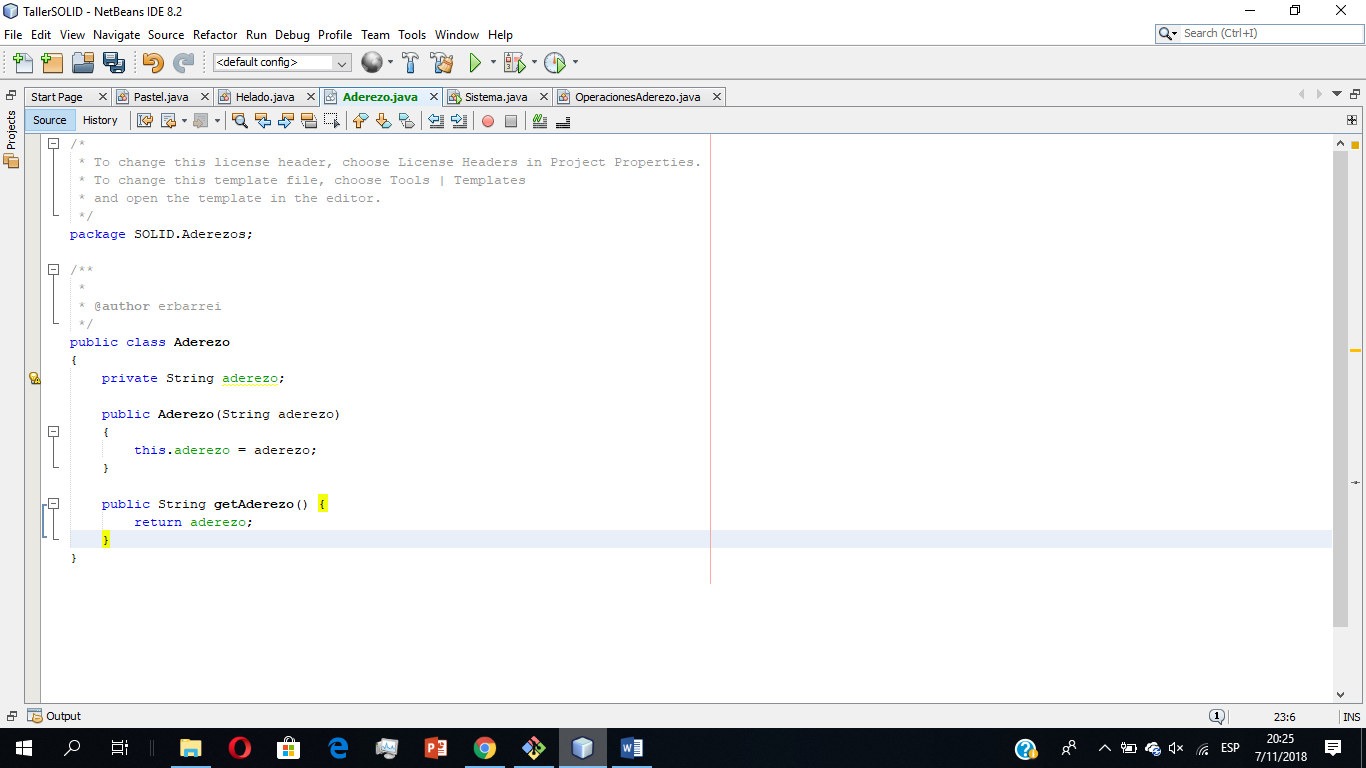
En la clase Helado y Pastel se está irrespetando el Liskov Subtitution principle, porque son clases parecidas y pueden ser reemplazadas por una clase padre





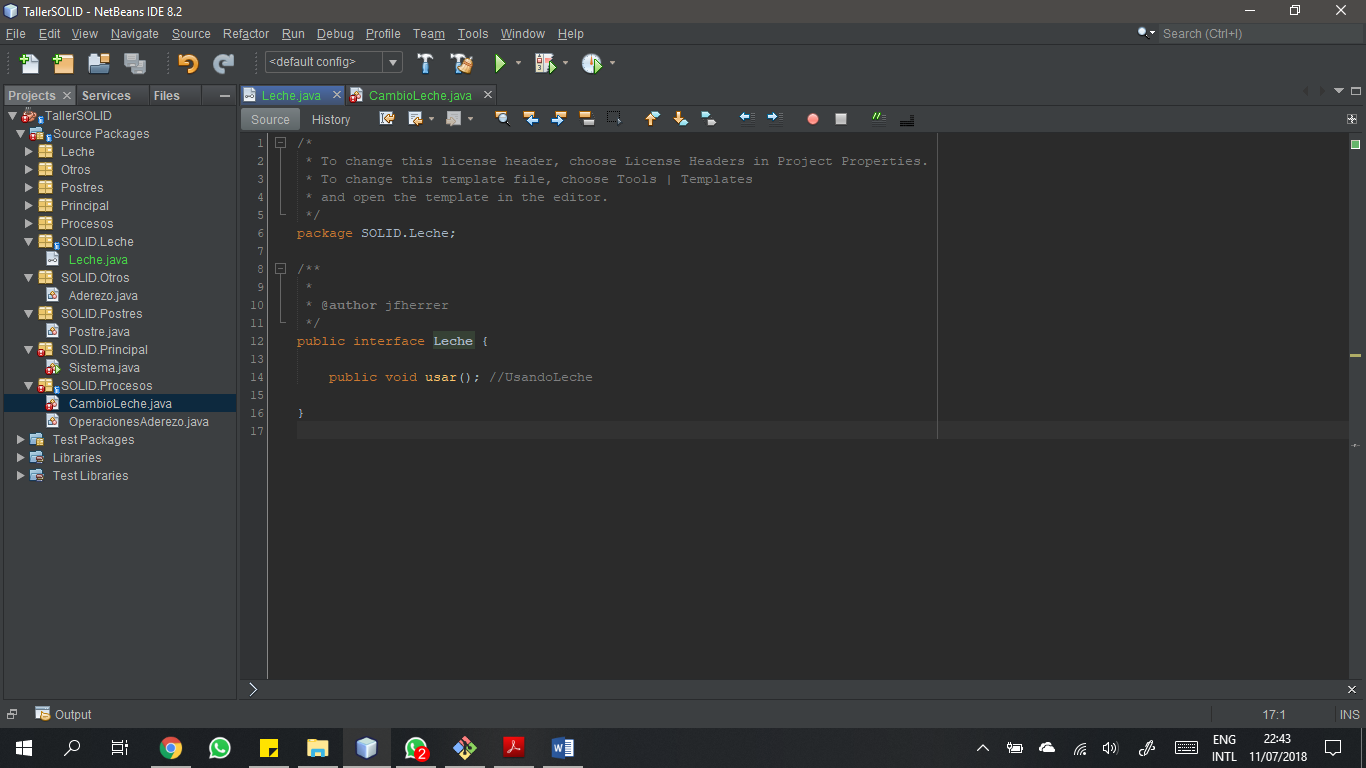
(Solución del primer principio violado)Se creó una nueva clase llamada Postre, la cual tendrá como atributo un postre en específico (helado, pastel, etc). Y también se modificó la clase OperacionesAderezos, para hacerla más general y agregar o remover aderezos de un postre y no de un postre en específico

2.



Aquí se incumple el principio de Open Closed, porque cada vez que querramos más tipos de aderezos, debemos modificar el enúm aderezos. Se creó una clase aderezo ya que, si se quieren añadir otros aderezos que no se encuentren en el enum, tendremos que crear uno nuevo. Esto se puede arreglar creando una clase abstracta aderezos y todos los aderezos serán sus clases hijas pero esto violaría el principio de responsabilidad única así que lo q se hizo fue crear una clase Aderezo que tiene en el constructor el tipo de aderezo que se desea utilizar.

3.



Se creó la interfaz Leche, la cual tiene un método que será instanciado por los diferentes tipos de leche que el usuario requiera. Aquí se violaba el principio de Single Responsibility, debido al código repetido en usar(), además de que también violaba el principio Open Closed, porque como se dijo usaba el mismo método en varias clases