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.

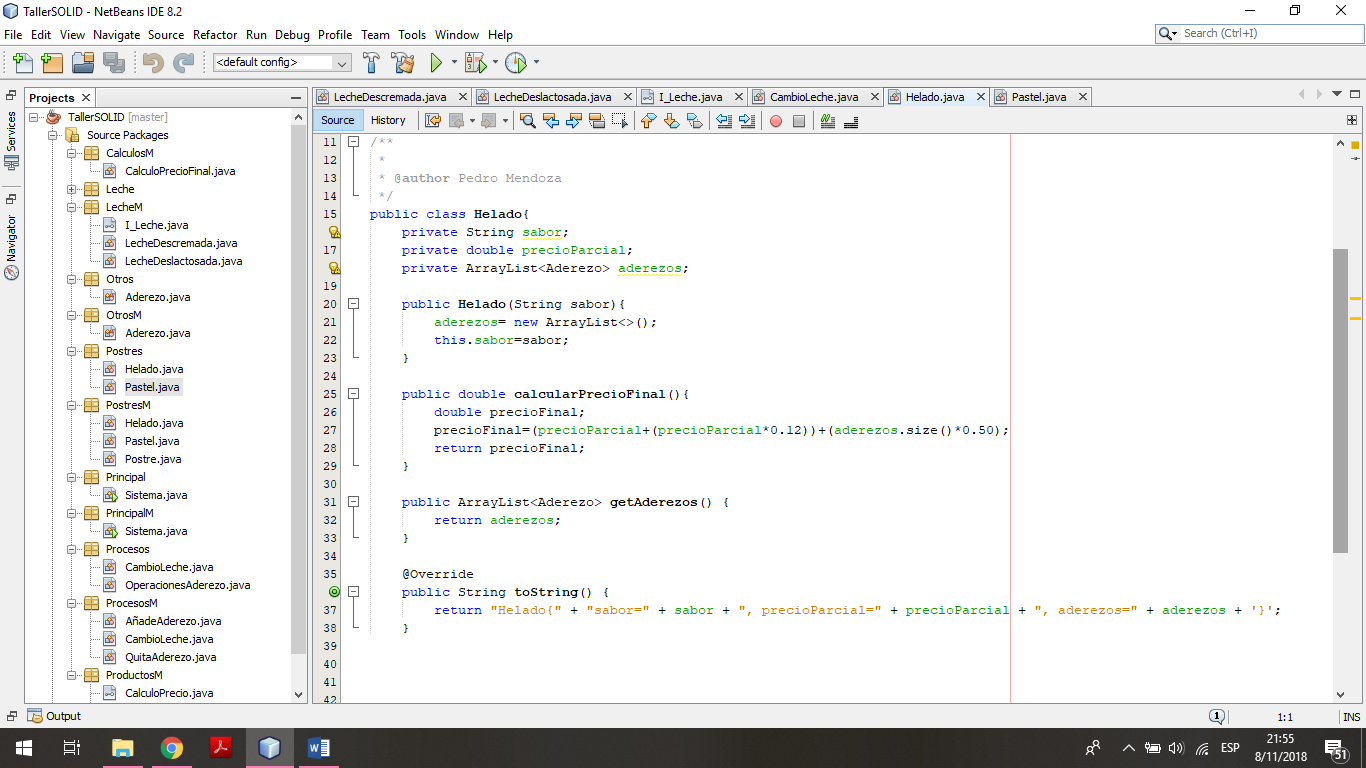
# Desarrollo

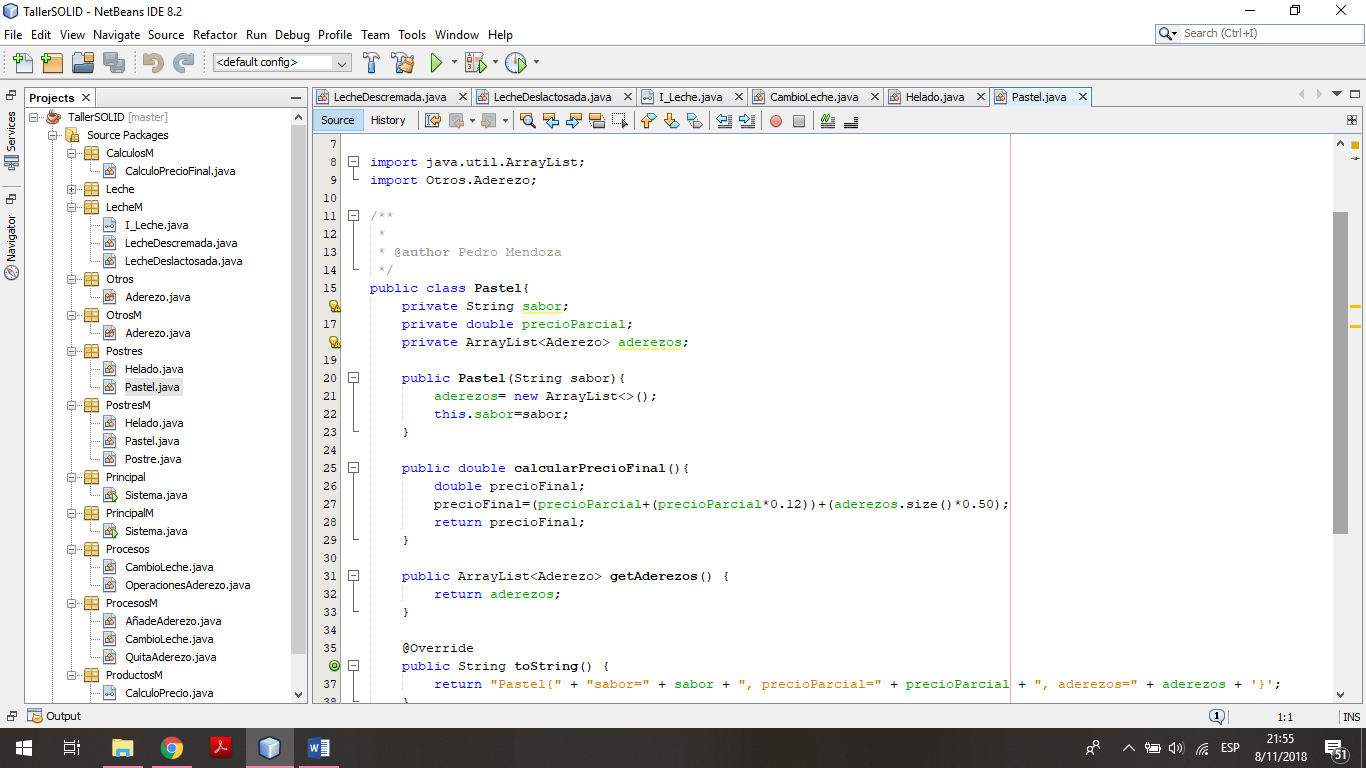
Los principios Solid que no han sido respetados son:

* **Single-Responsibility Principle:** 
  + Existe repetición de atributos y métodos en las clases pastel y helado como sabor, precio parcial y aderezos además del método calcularPrecioFinal() y getAderezos().
  + En la clase de OperacionesAderezo existe un método para agregar aderezos y otro para quitarlos por lo que no posee una responsabilidad o un solo trabajo.
* **Interface Segregation Principle:** 
  + En las clases leche descremada y leche deslactosada existe un solo método que es usar el cuál no ha sido implementado y tiene el mismo funcionamiento para cada tipo.
* **Open-Close Principle:** 
  + En la clase Operaciones Aderezo, se agregan y quitan aderezos dependiendo del tipo de producto que vendan, por lo que, si se modifica o se agrega un producto, el sistema este forzado modificar gran parte del código.
  + Si se agregan nuevos productos como bebidas se duplica código para calcular el precio final de un producto.
* **Dependecy Inversion:**
  + En la clase CambiarLeche se creaban los objetos internamente en las funciones que tiene dicha clase, se precedió a pasar por parámetro los objetos leche y posteriormente a usar polimorfismo para asegurar que el principio open/close tampoco sea violado por la modificación realizada.

**Implementación:**

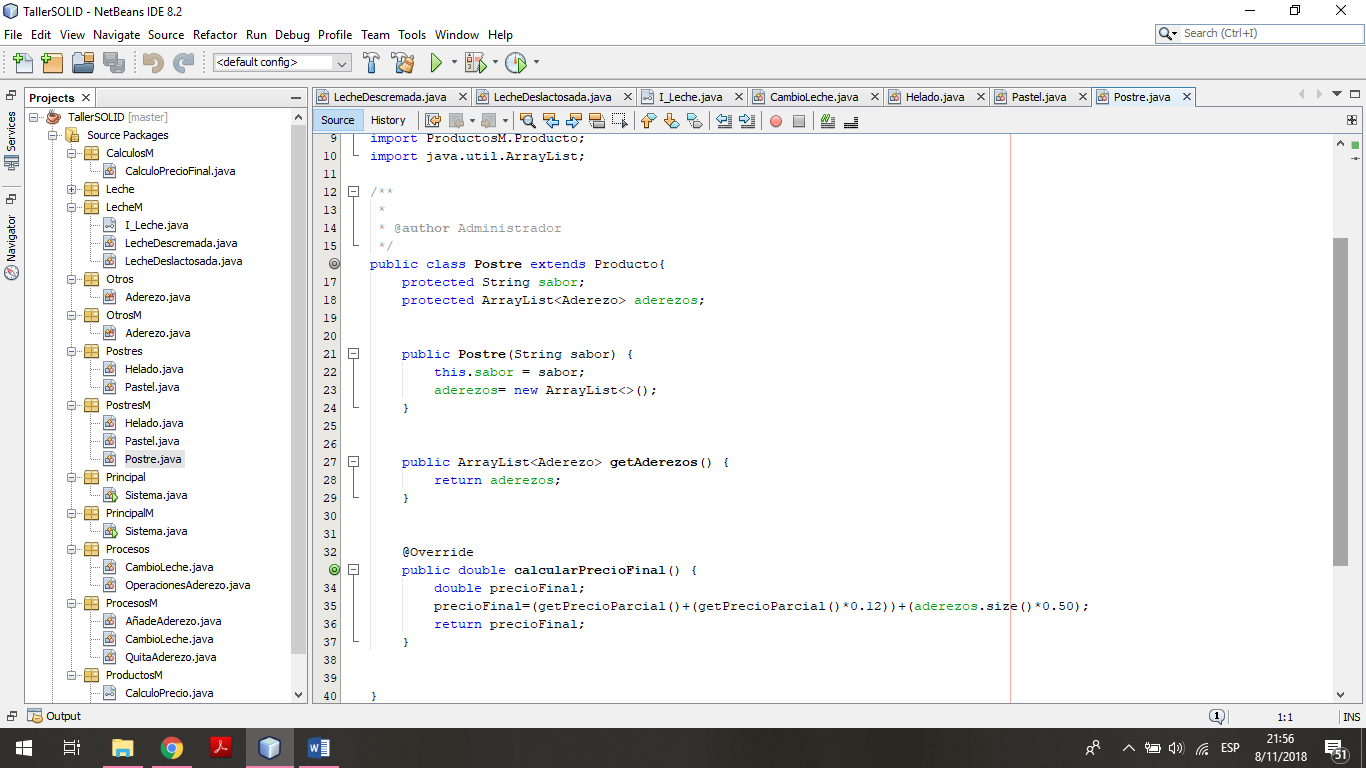
SINGLE-RESPONSIBILITY PRINCIPLE



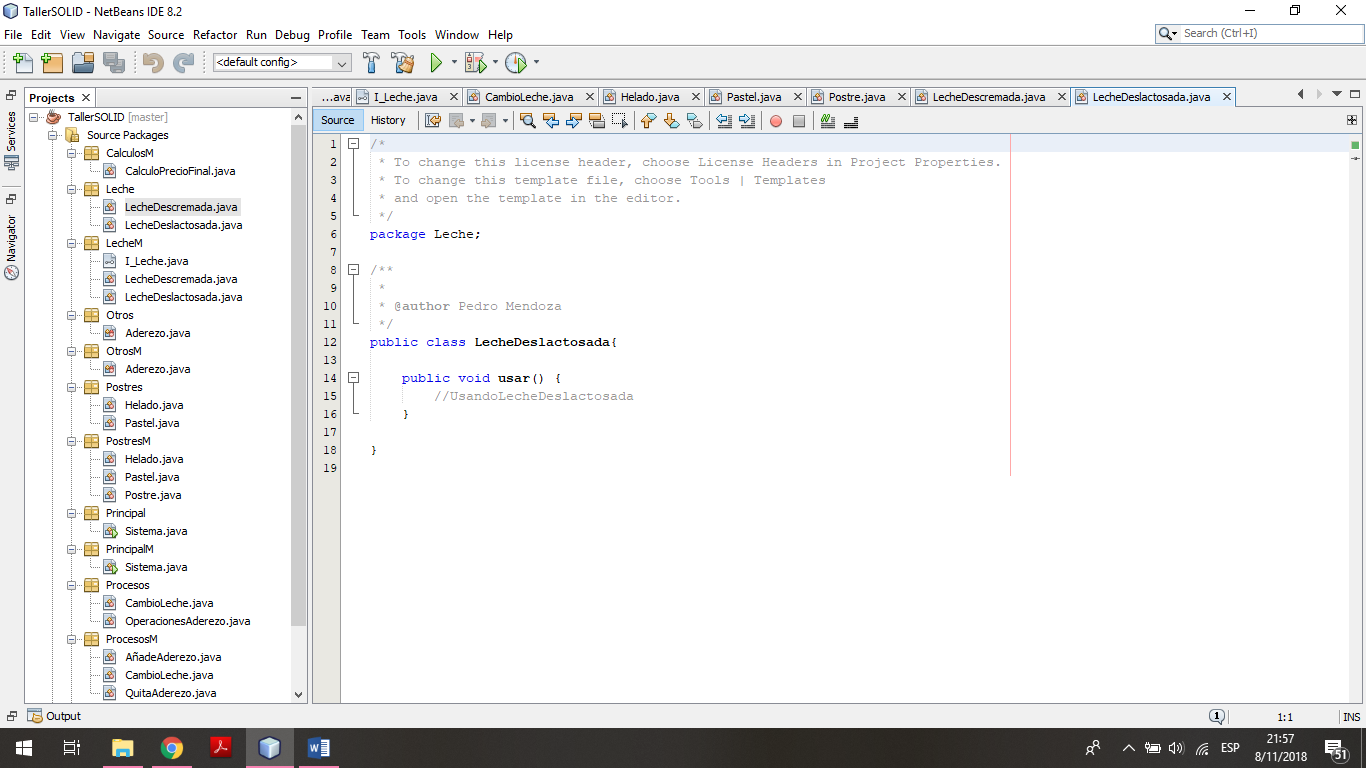
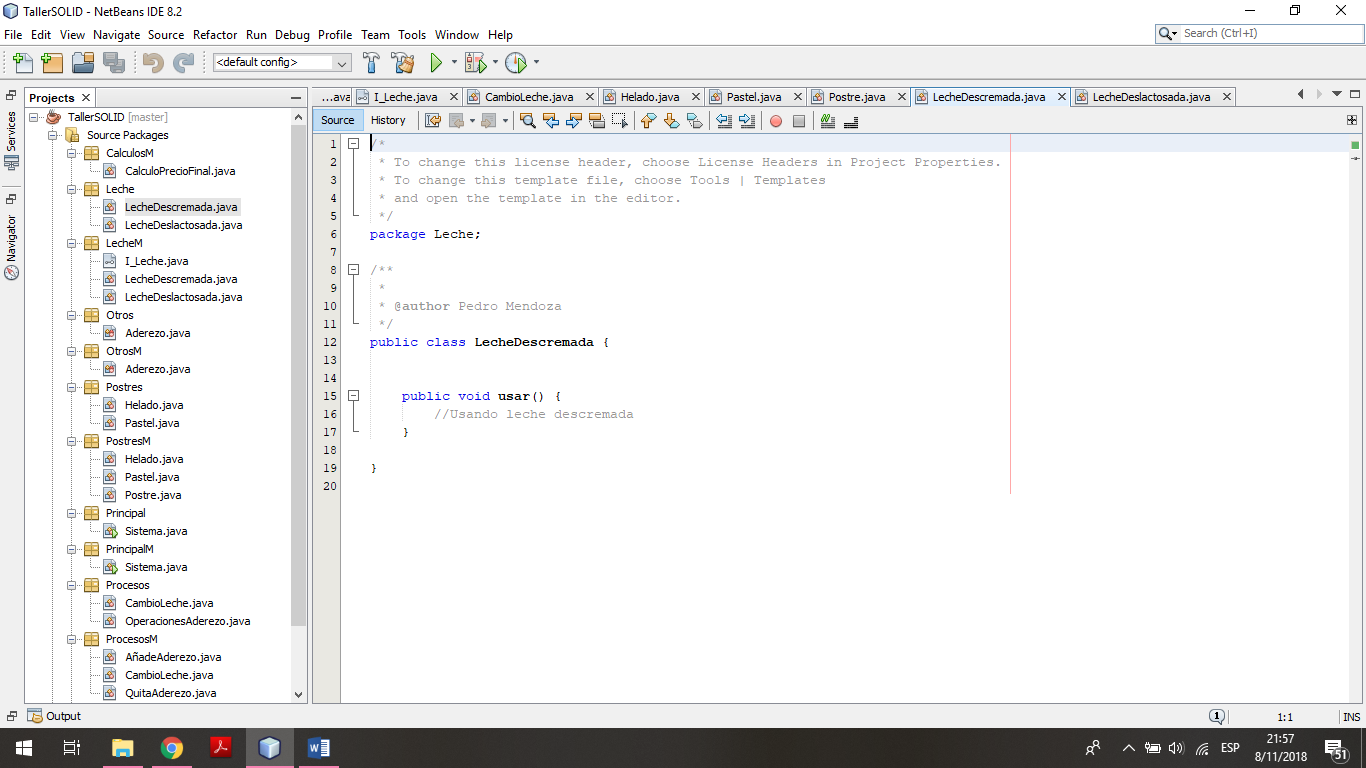


Solución:

Se crea una clase general llamada Postre, con los atributos repetidos.

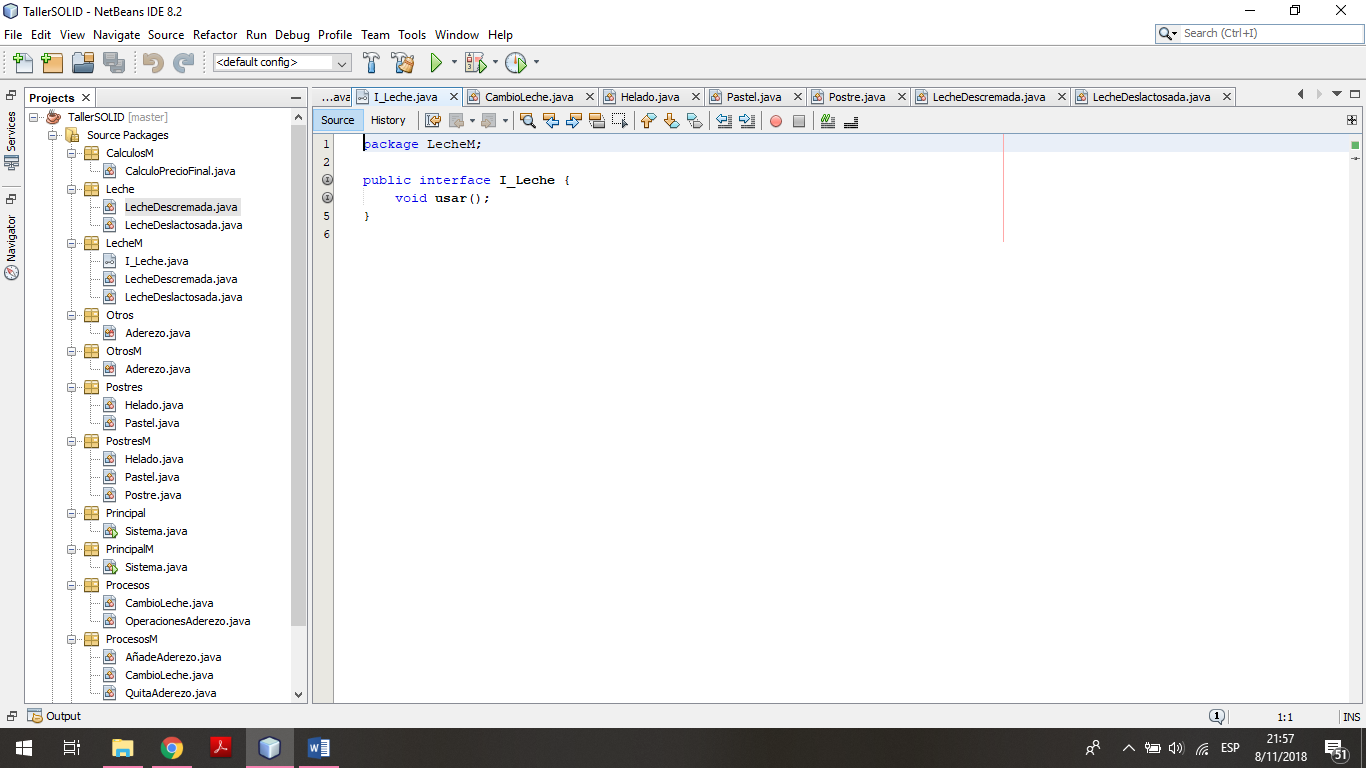


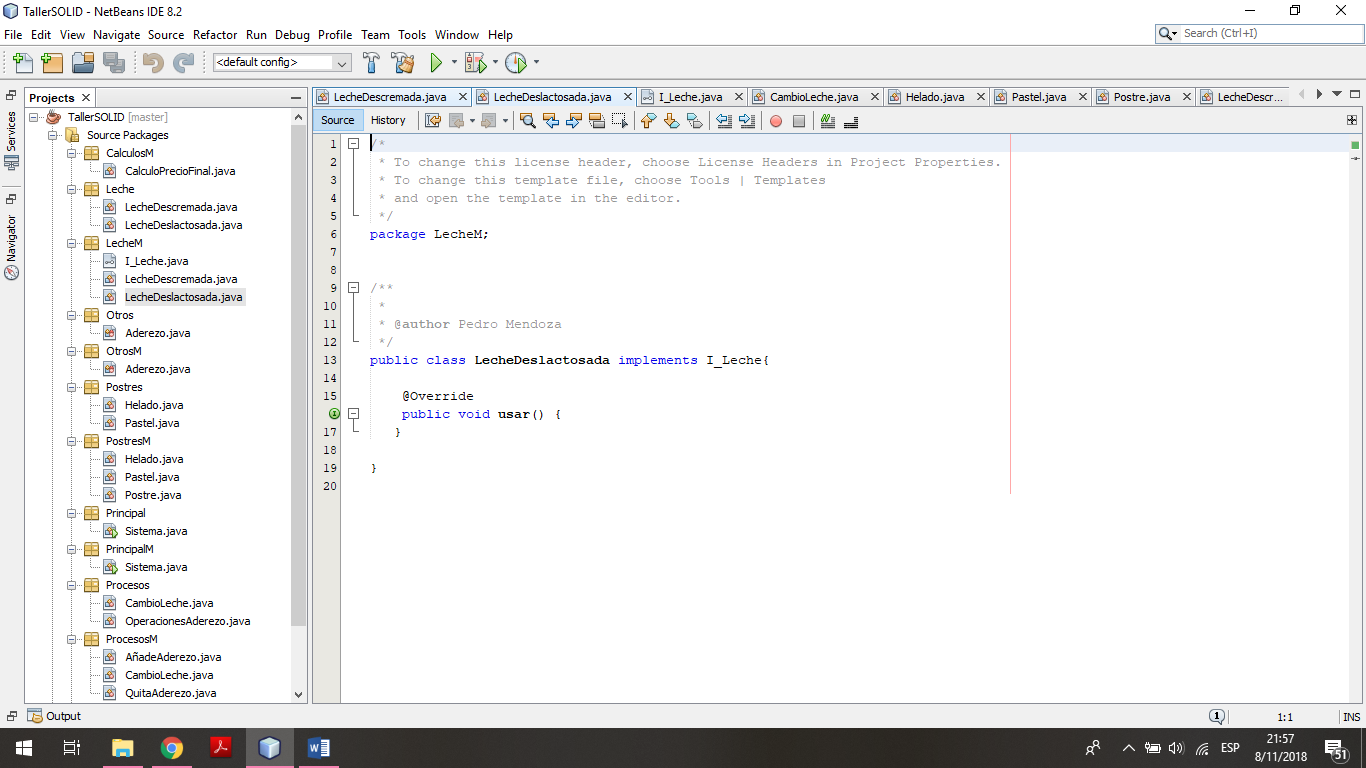
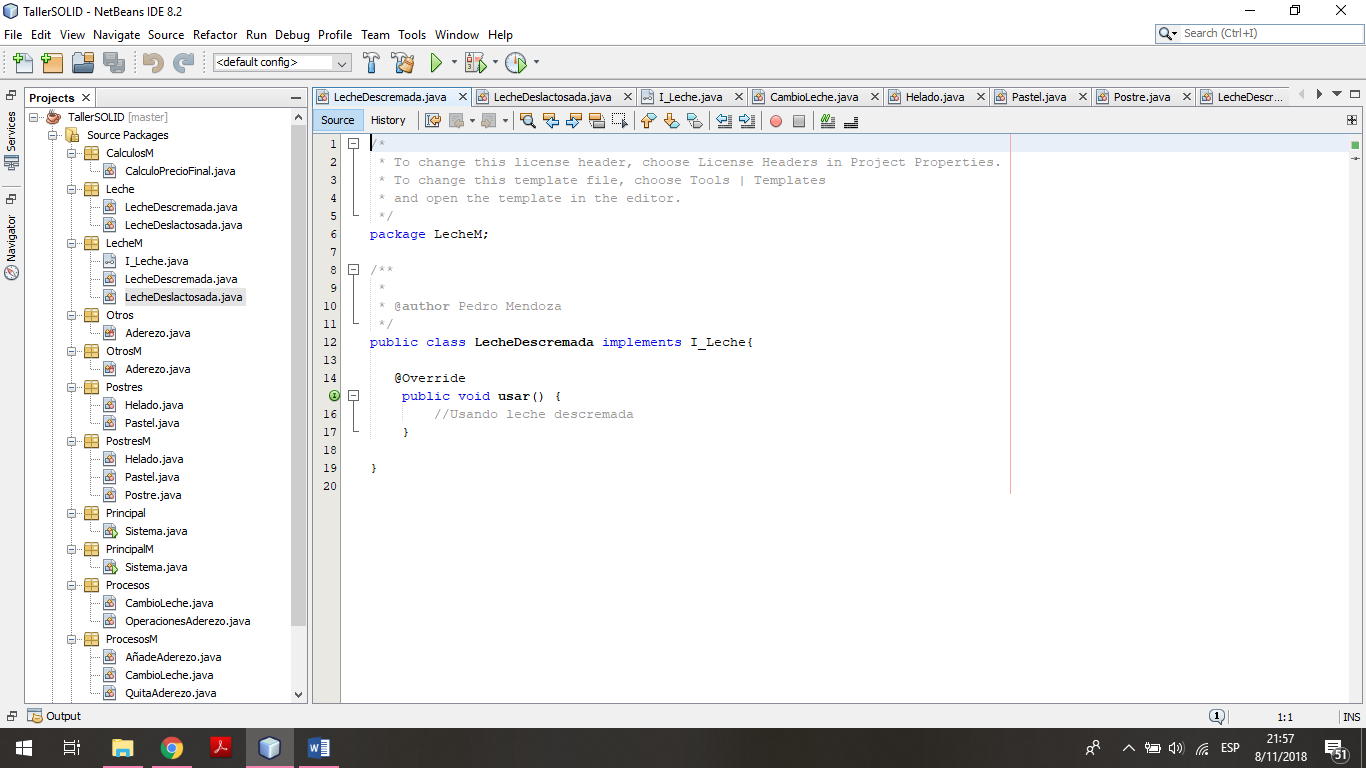
INTERFACE SEGREGATION PRINCIPLE



Solución:

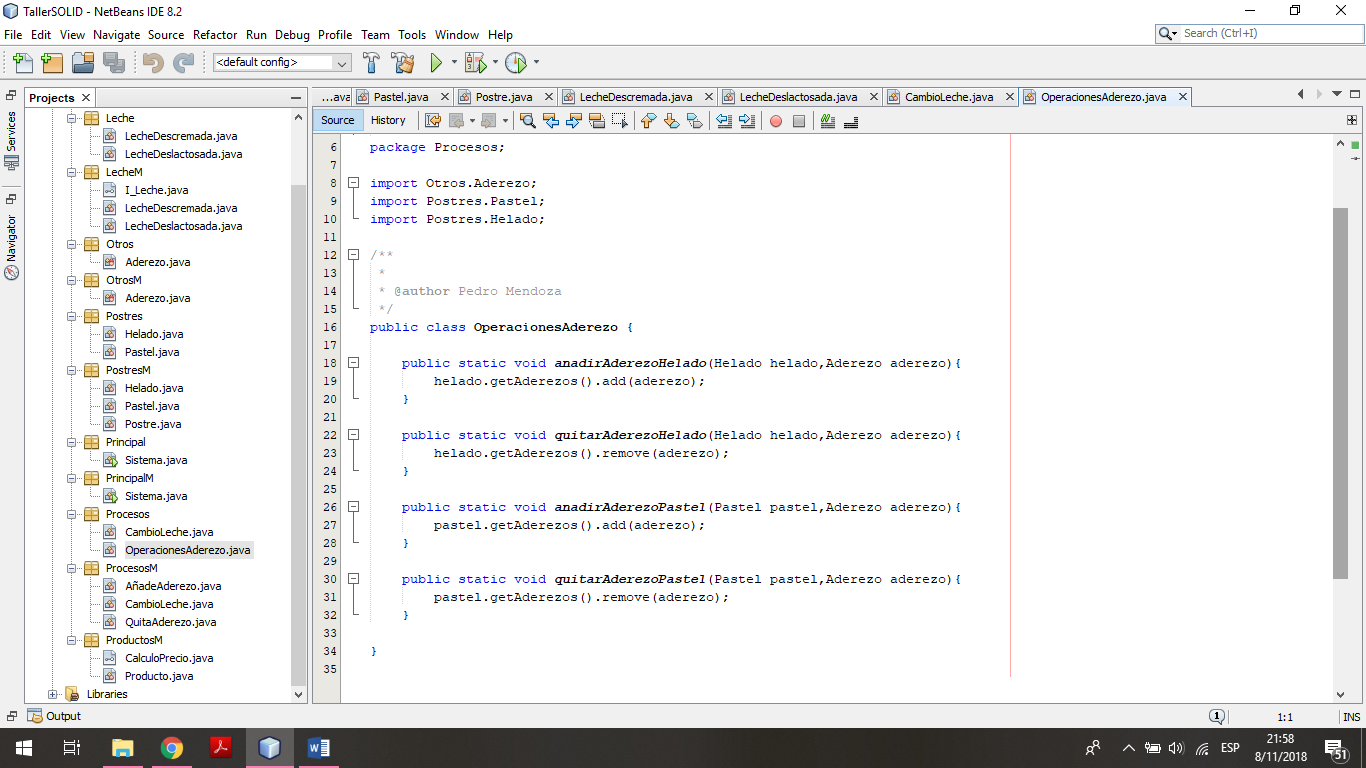
Se crea una interfaz para que sea implementada por cada tipo de leche existente en el negocio, la cual contenga la función usar().



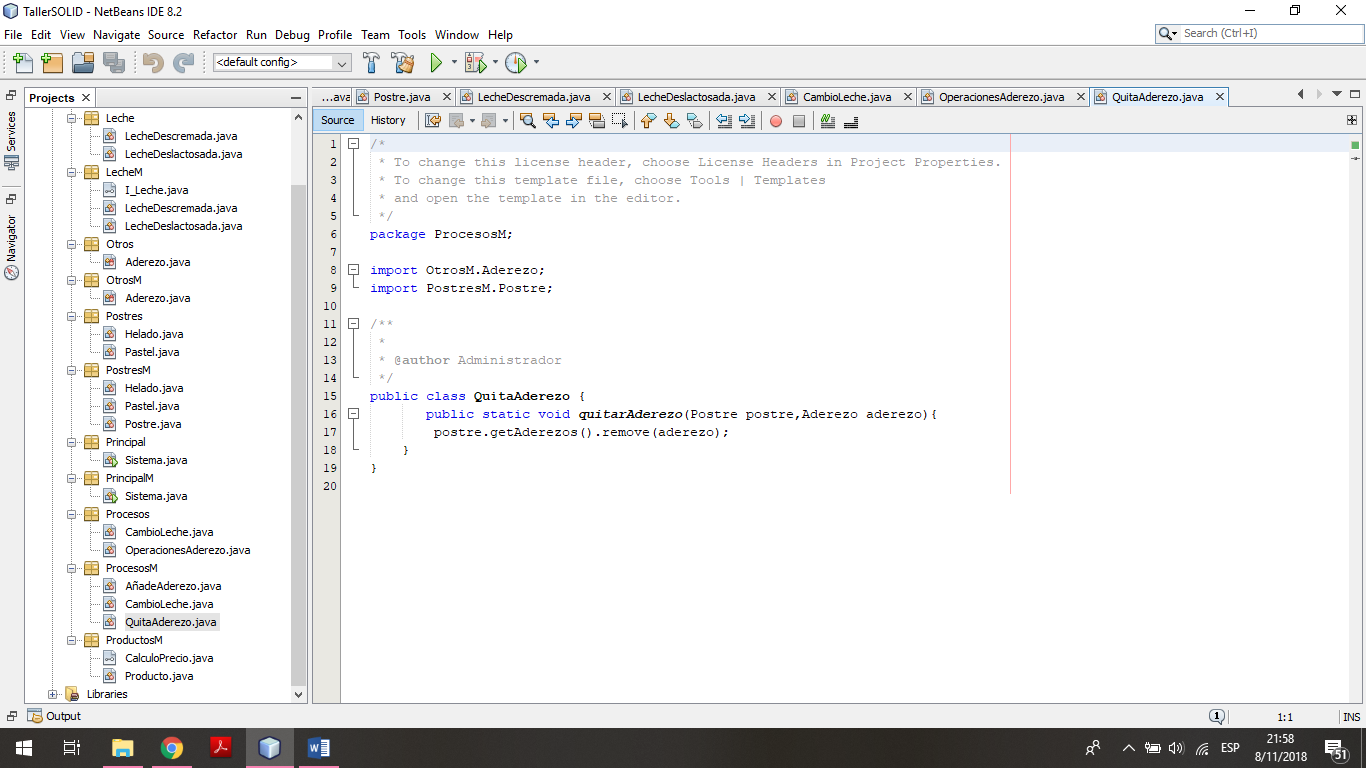
OPEN-CLOSE PRINCIPLE

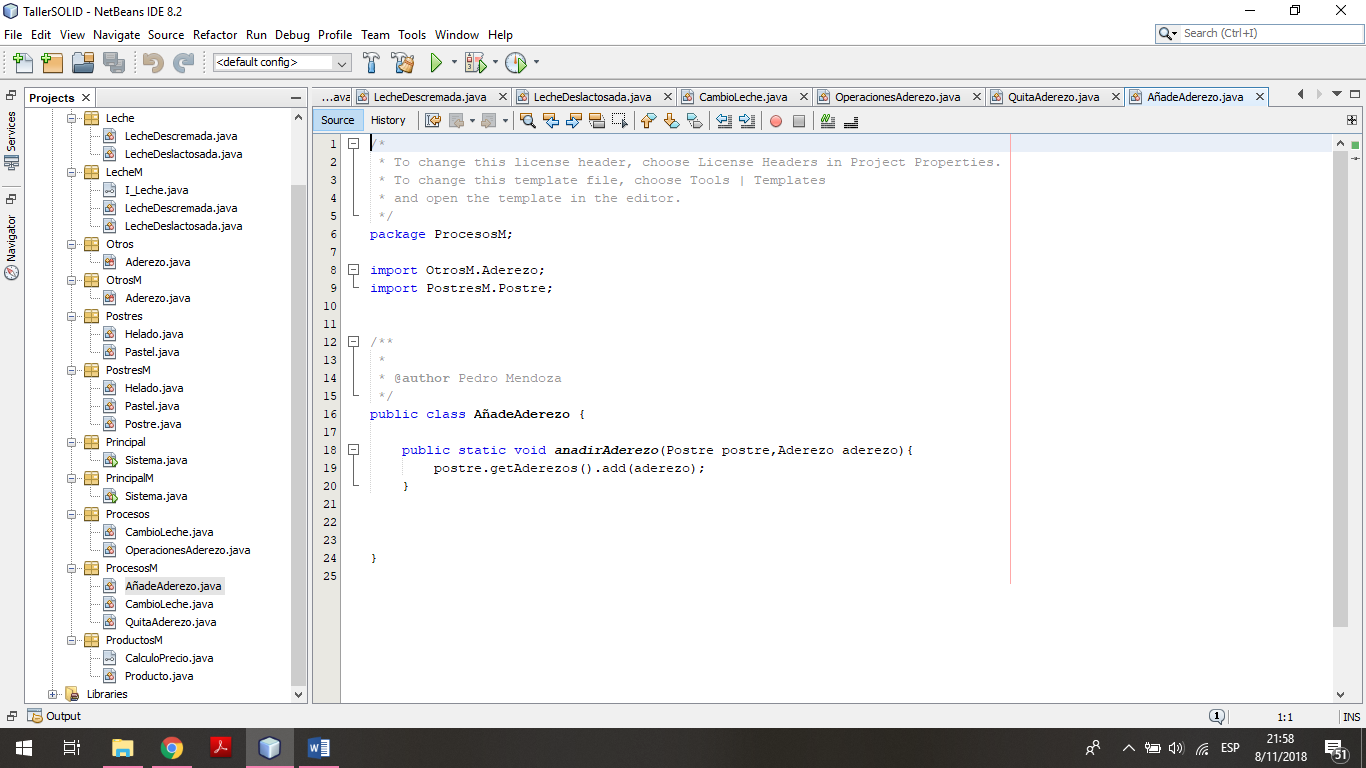
**Problema 1:**



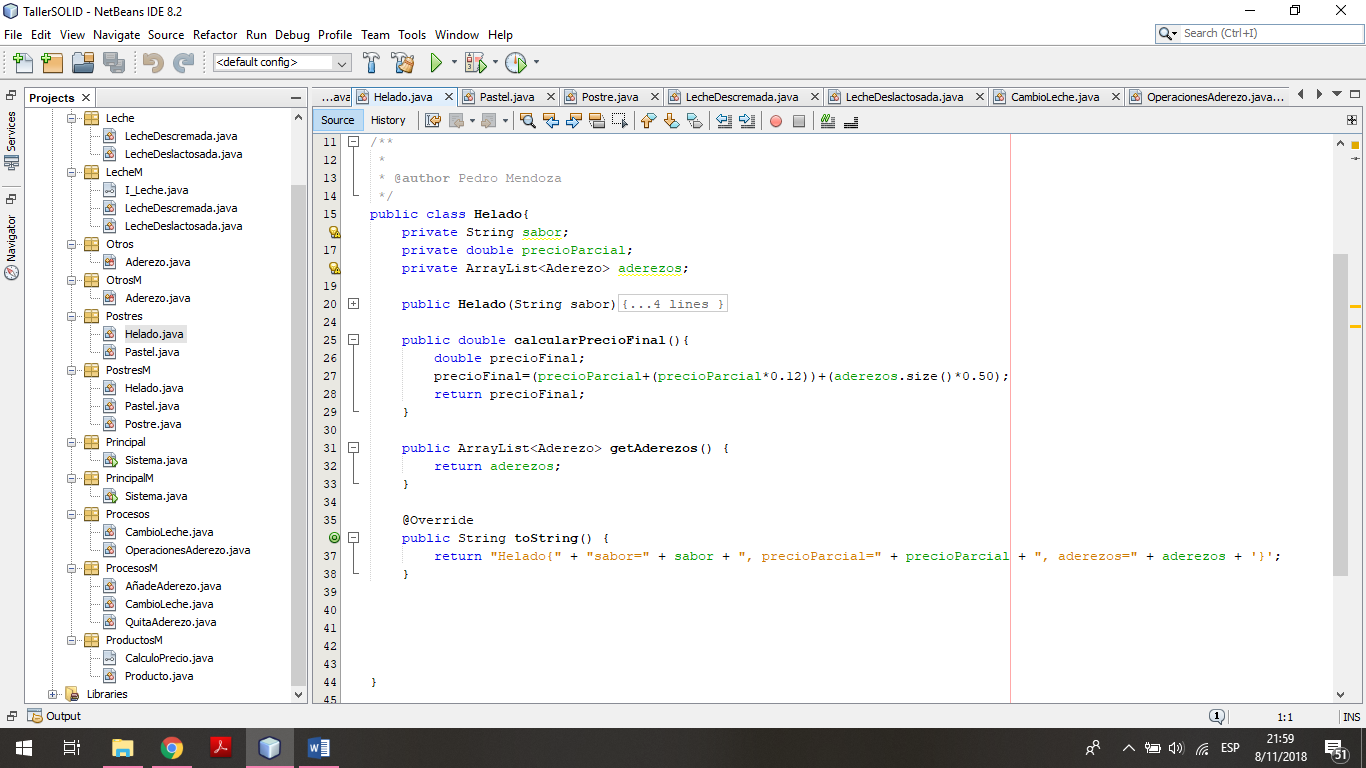
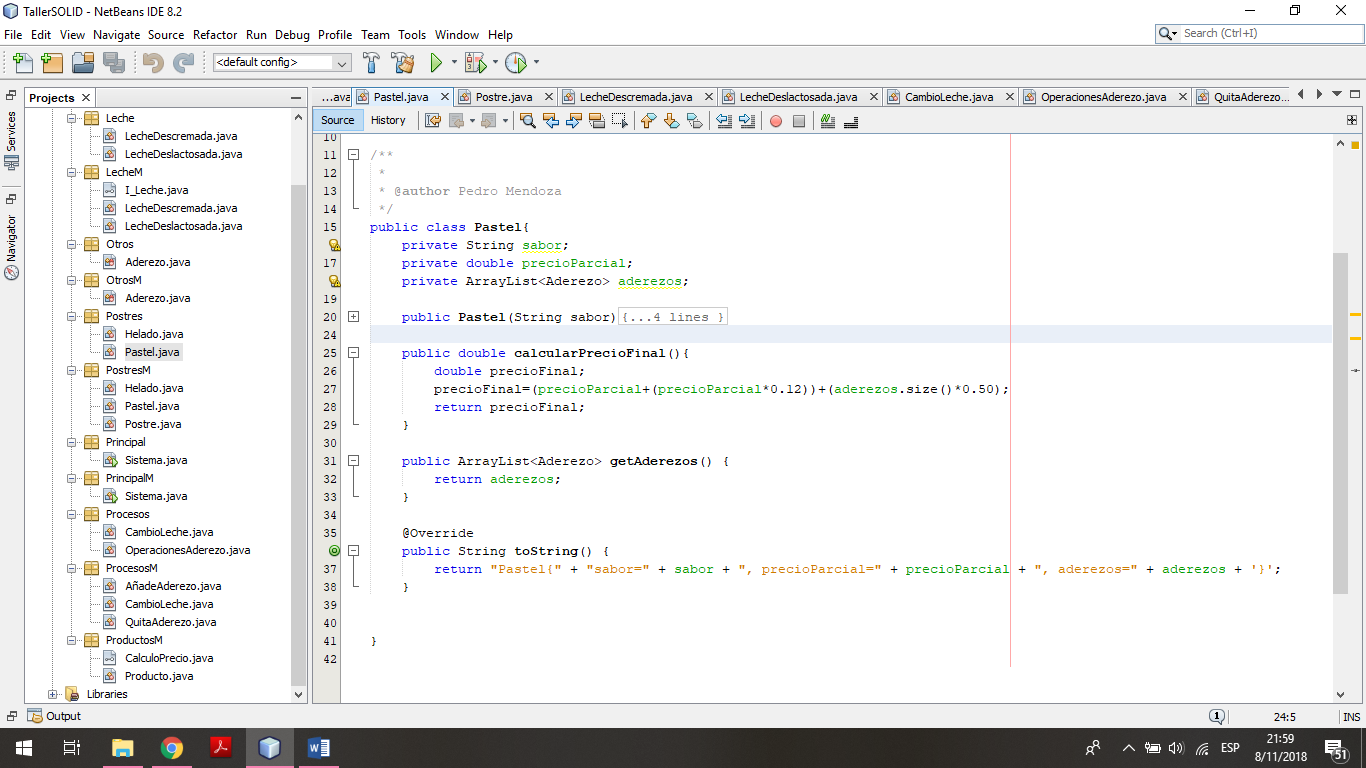
Solución:

Se crea un solo método tanto para agregar como para quitar aderezo, aceptando como parámetro la clase padre Postre, ya que posee los métodos que la función utiliza. No obstante, esto incumple el principio de Single-Responsibility, por lo que se crea una clase para cada función.



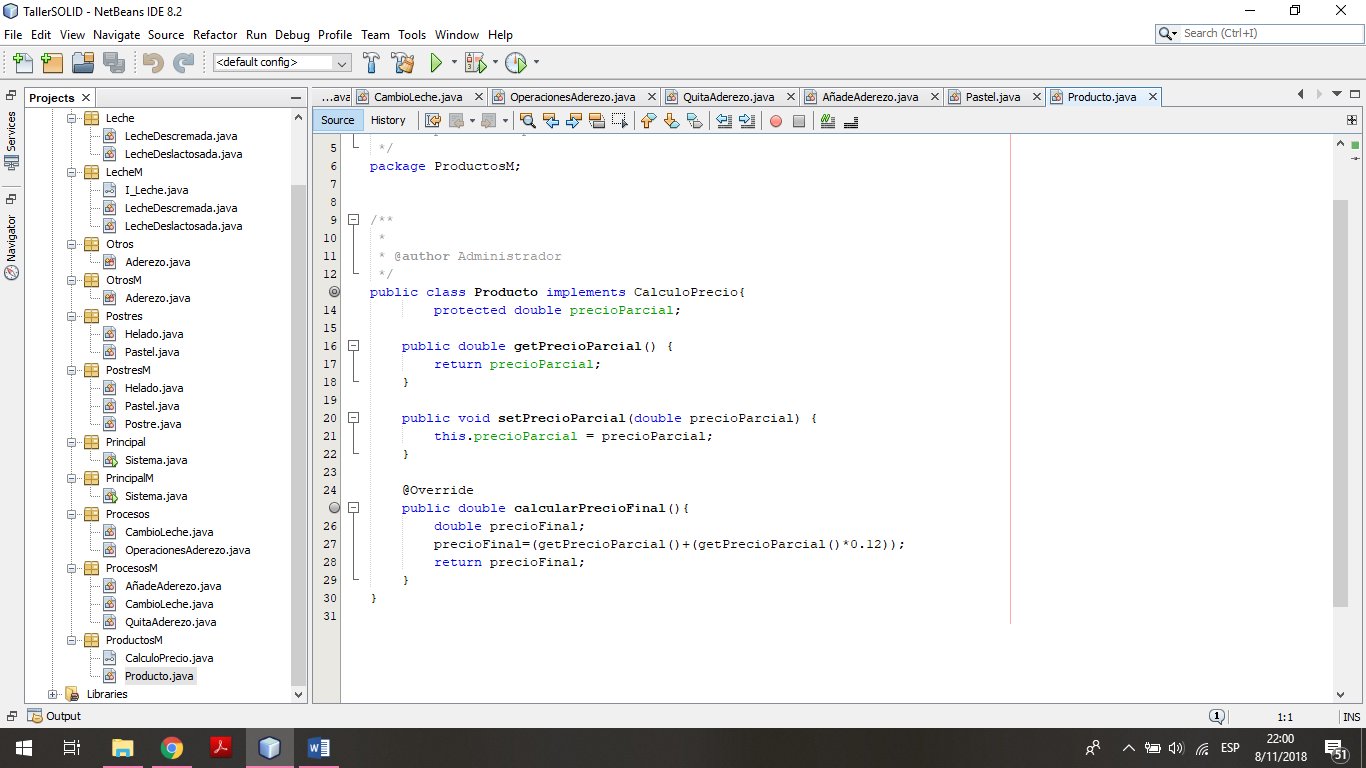


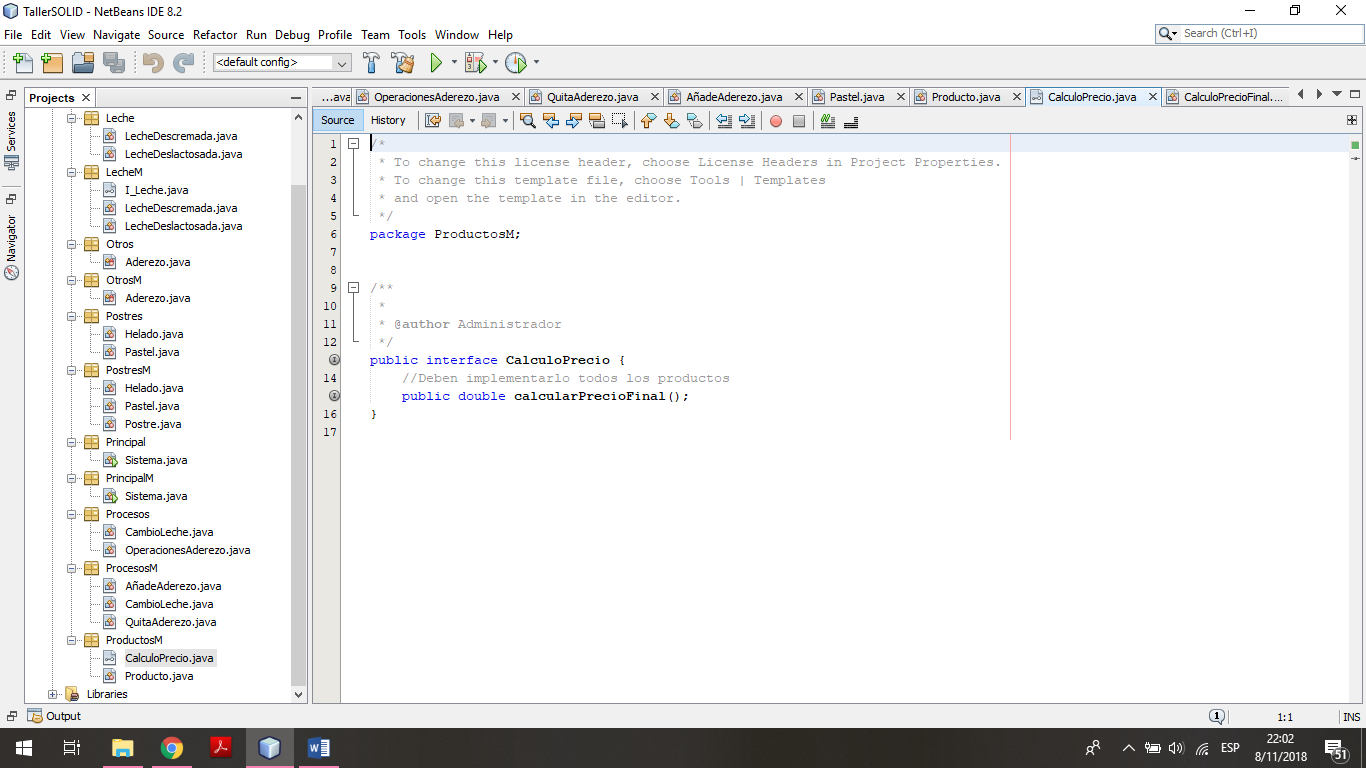
**Problema 2:**

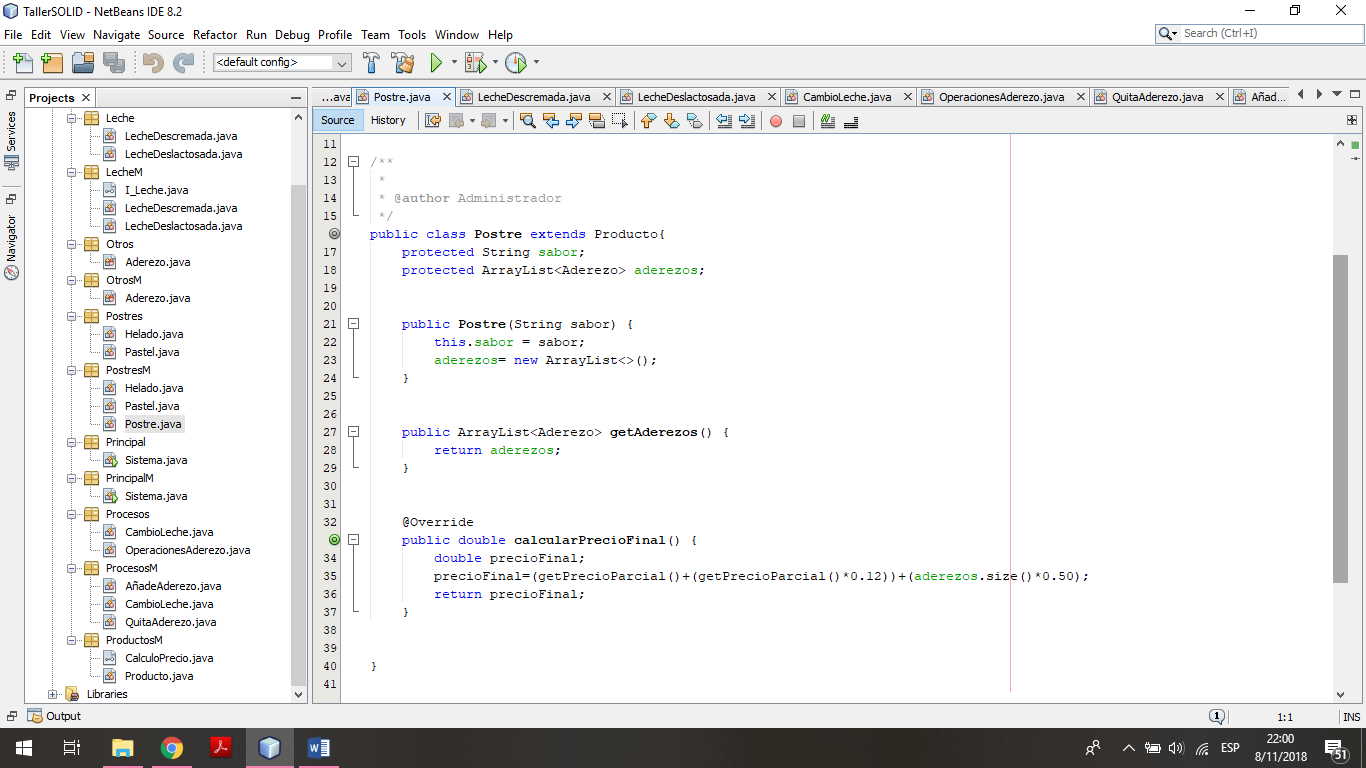


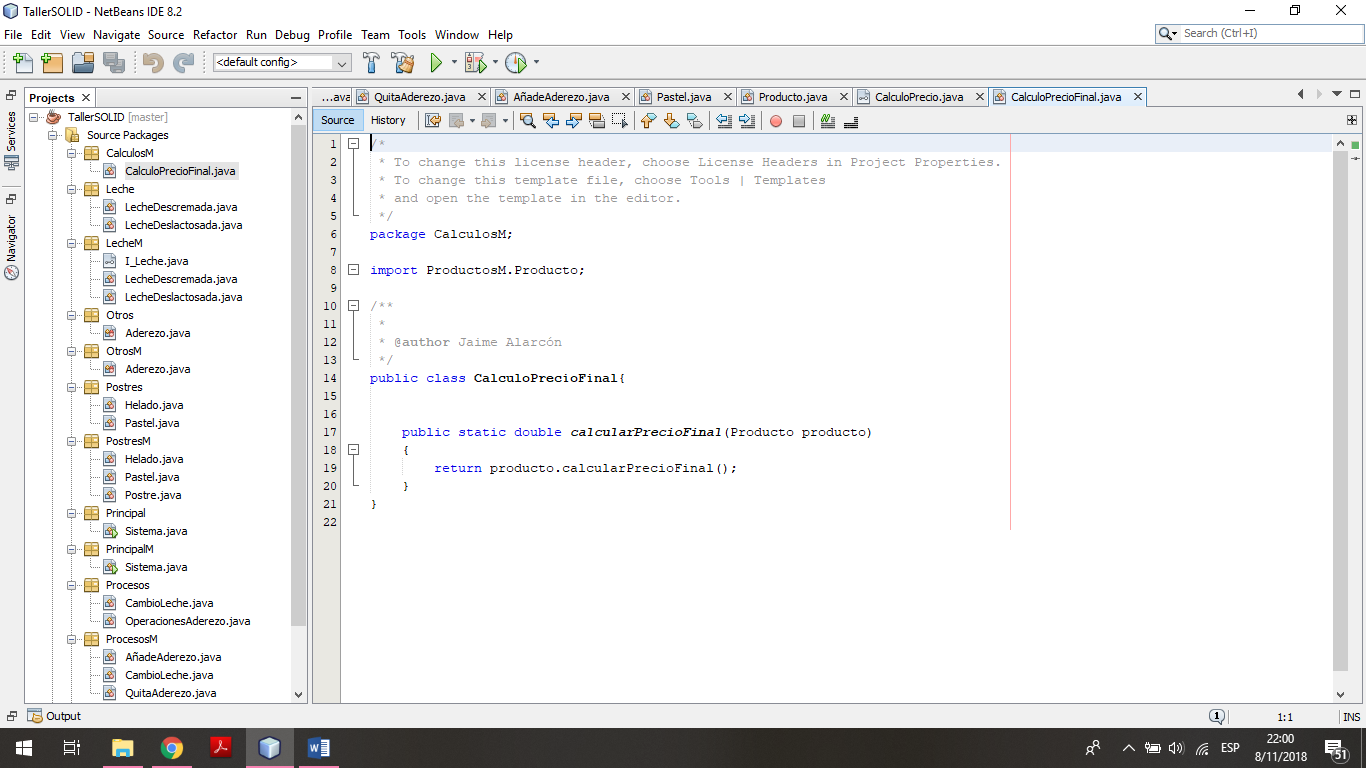
Solución:

Se crea una clase general de los productos del negocio ya que el precio final de cada uno de ellos tiene como base el precio parcial y el porcentaje de IVA que depende del mismo atributo por lo tanto, el método CalcularPrecioFinal() se lo implementa en esta clase, pero como este método cambia cuando el producto es un postre, se sobrescribe este método en la clase respectiva. Sin embargo, esto viola el principio de Single-Responsibility, entonces se crea una clase externa que solo se encarga del cálculo del precio Final, el cuál retorna la salida del método implementado en cada una de las clases mencionada. Además, para no violar el principio de Interface Segregation se crea una interfaz de cálculo ya que todos los productos deben implementar este método.

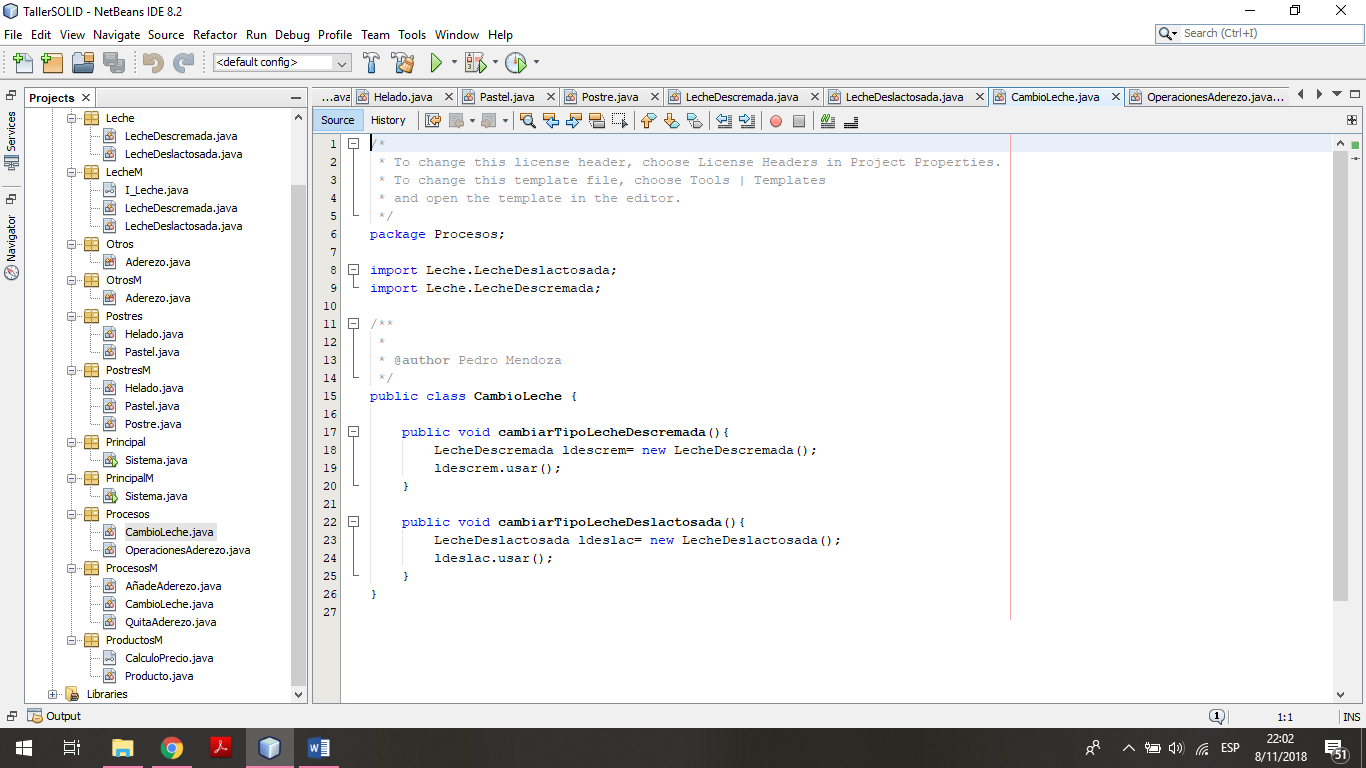








DEPENDECY INVERSION



Solución:

Para no instanciar nuevos objetos dentro de métodos, aquellos se envían como parámetro en este caso se envía el tipo de leche como parámetro de los métodos CambiarTipoLecheDescremada() y CambiarTipoLecheDeslactosada().

