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

Joel Espinoza, Kevin Celi

**1.-**

* **Principio de Inversión de Dependencias**

Lo que rompe el Principio de Inversión de Dependencias se puede observar en el paquete de operaciones, en la clase de cambio de leches, en el cual nos vemos en la necesidad de Inicializar una clase para poder usar su método dentro de ese método.

* **Principio de segregación de interfaces**

El principio de segregación de interfaces no está siendo respetado al no tener las interfaces implementadas para el cambio de leche en los postres, y tener creadas varias funciones para poder realizar los cambios.

Para solucionar este problema se realizo los siguientes cambios en las clases y se creó las interfaces.

* **Principio Open Closed**

Viola este principio en la clase de “Operaciones Aderezos” porque los métodos añadir y quitar aderezo se implementan para cada clase(Postre y helado) y si se quiere agregar una nueva forma se debe modificar el código, lo cual va en contra del principio.

* **Principio Single-Responsability**

Viola este principio debido a que una clase debe tener un solo trabajo, que en nuestro caso es crear un Objeto tipo Helado o a su vez un Objeto tipo Postre, pero agregarAderezo a un Objeto diferente al de la clase.

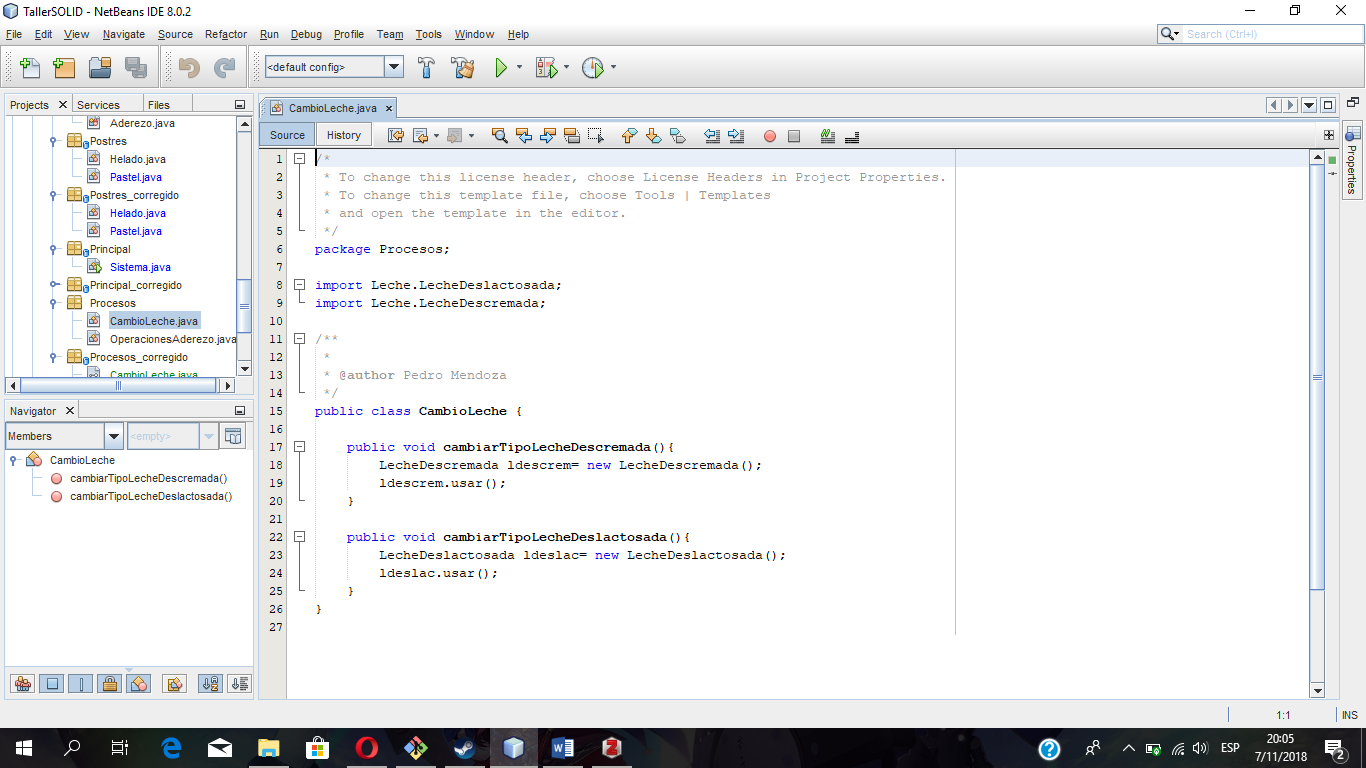
**3.-**

* **Principio de Inversión de Dependencias y segregación de interfaces**

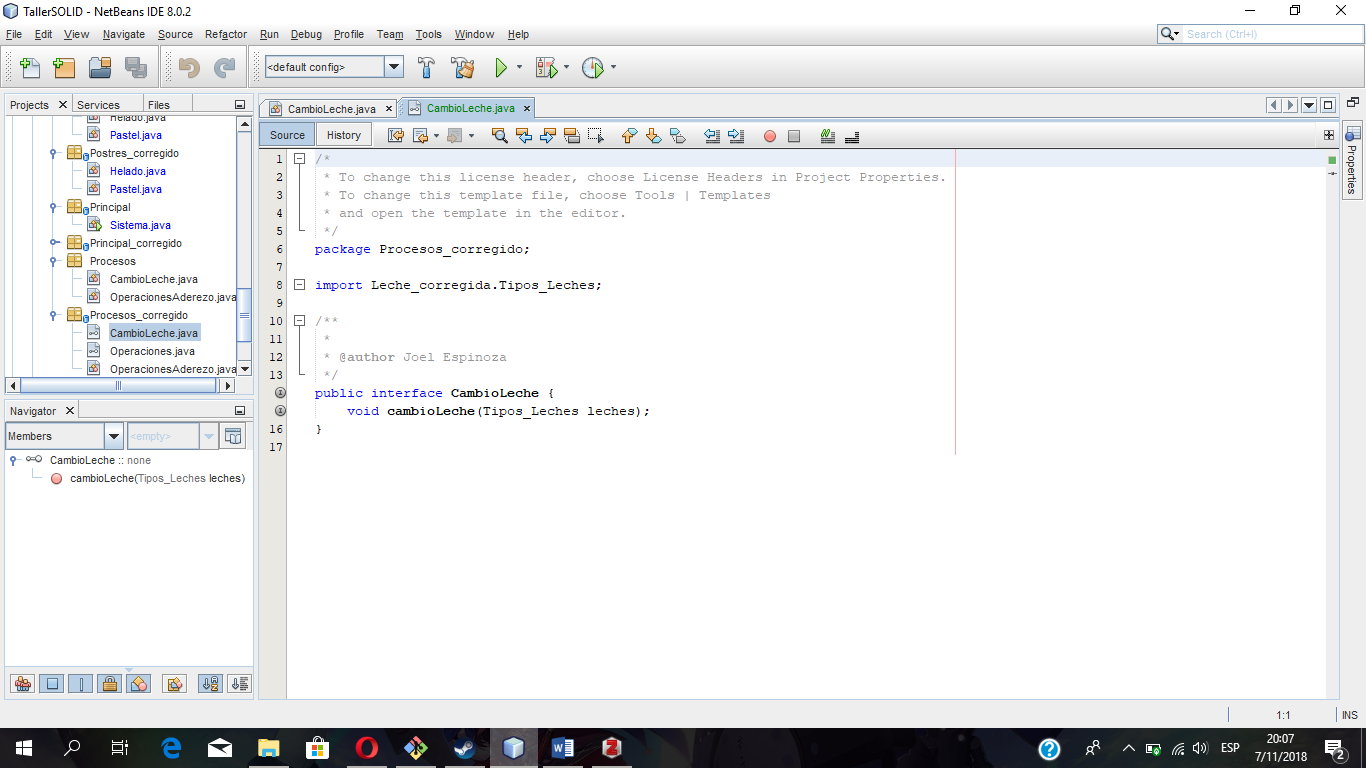
Para solucionar este problema se realizó los siguientes cambios en las clases y se creó las interfaces.

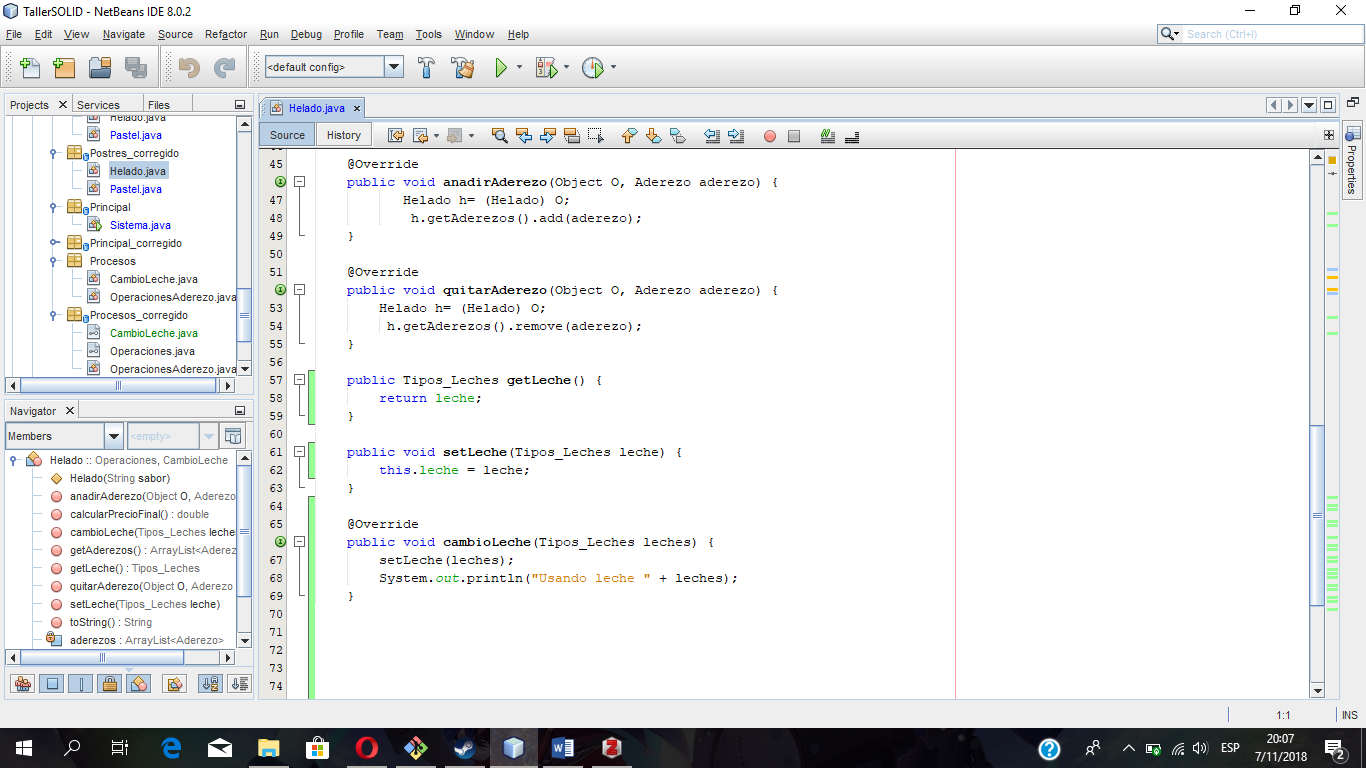
Lo que se hace es esa clase transformarla en una interfaz que se implementada dentro de las clases del paquete Postres.

Código que **rompe el principio** de inversión de dependencias y principio de segregación de interfaz



Código siguiente el principio de inversión de dependencias y principio de segregación de interfaz



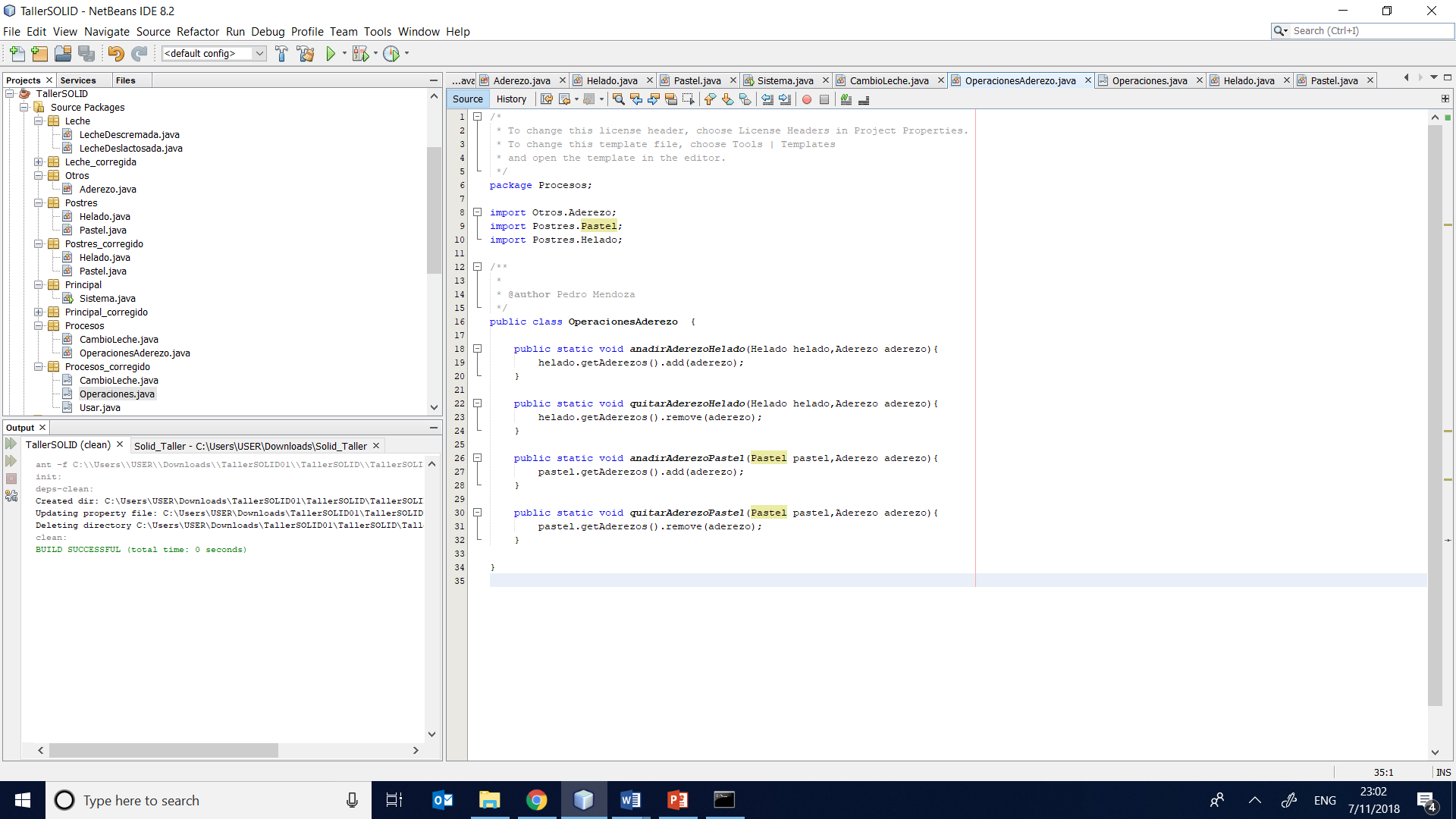
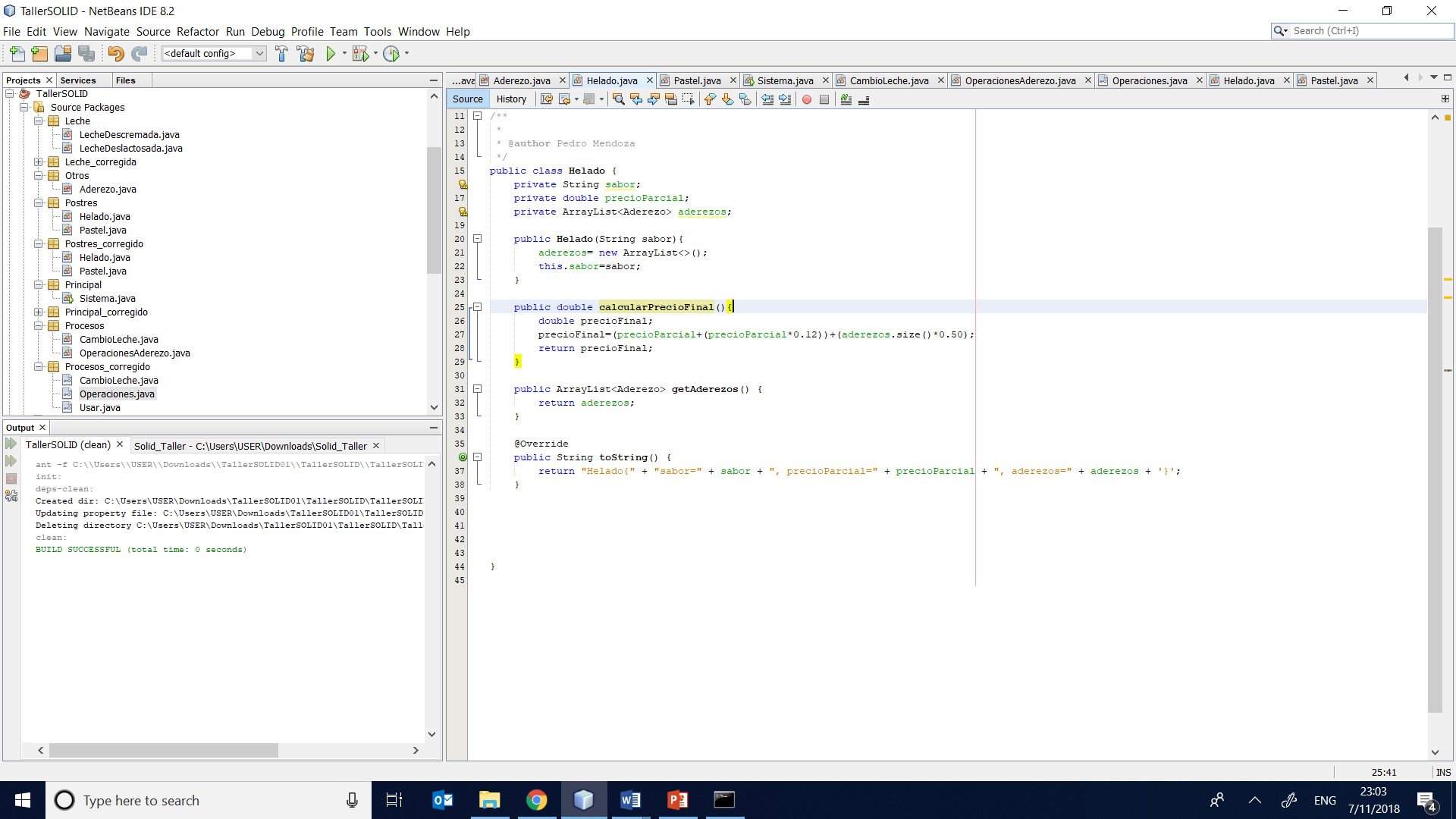
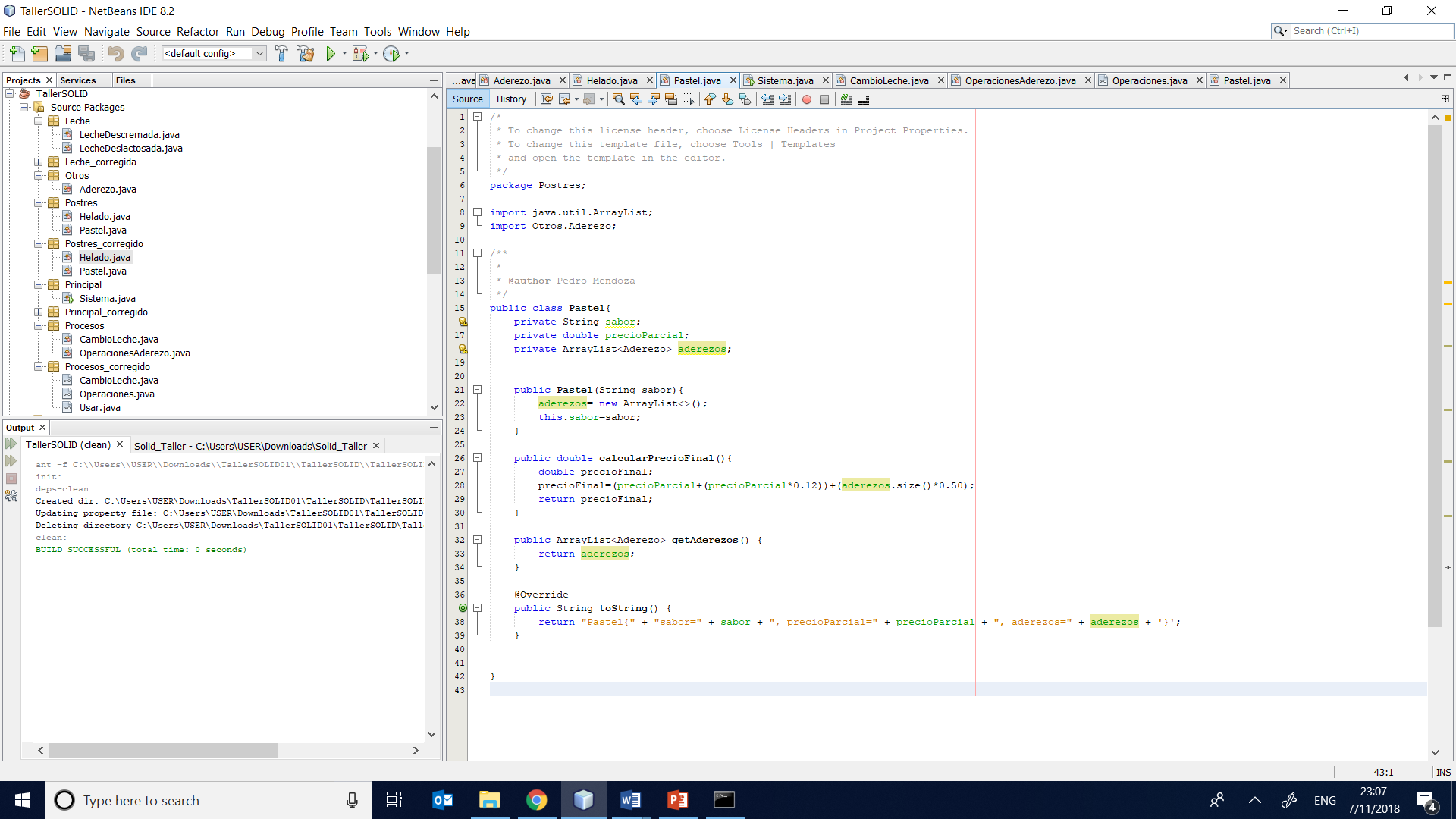


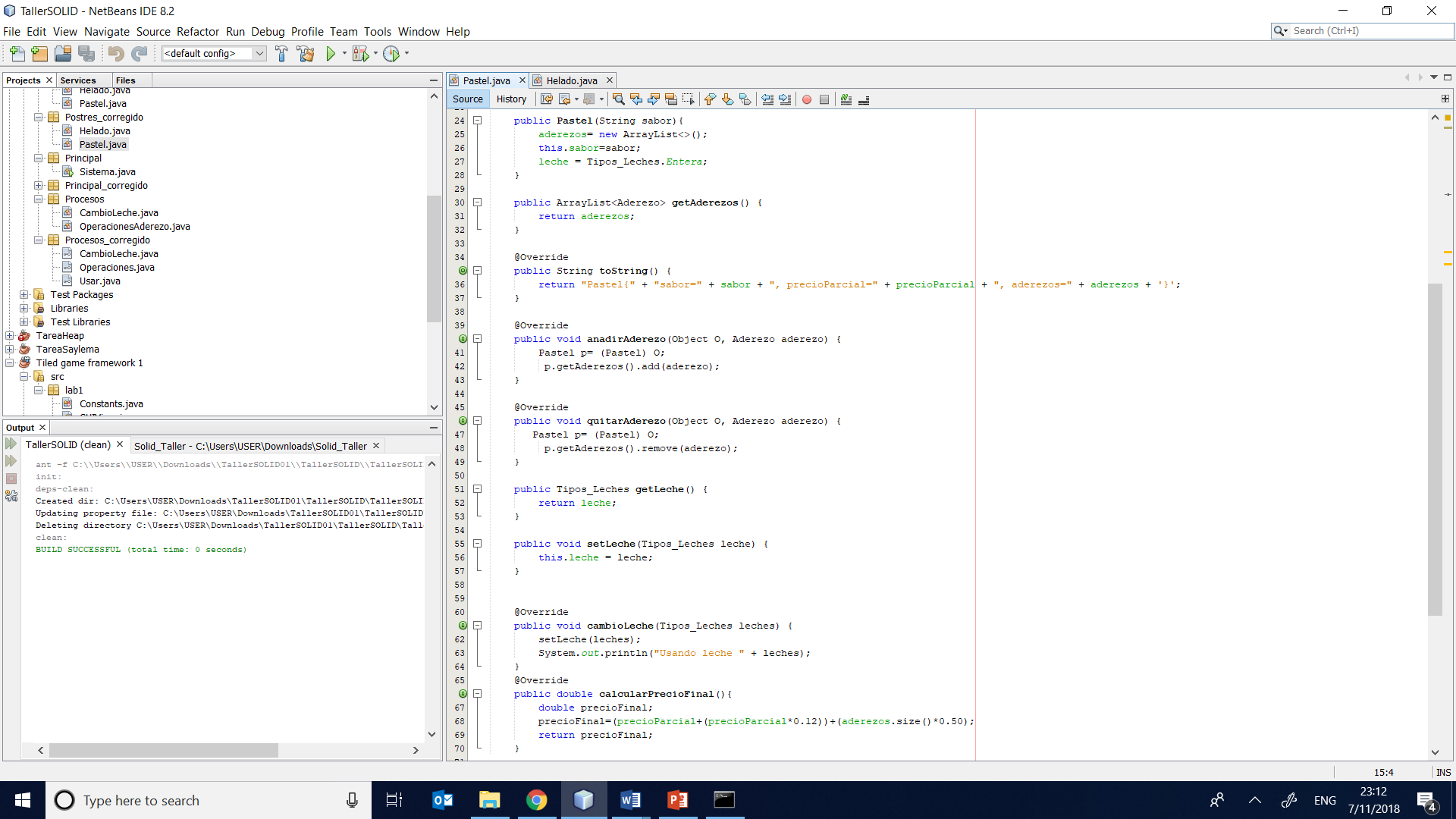
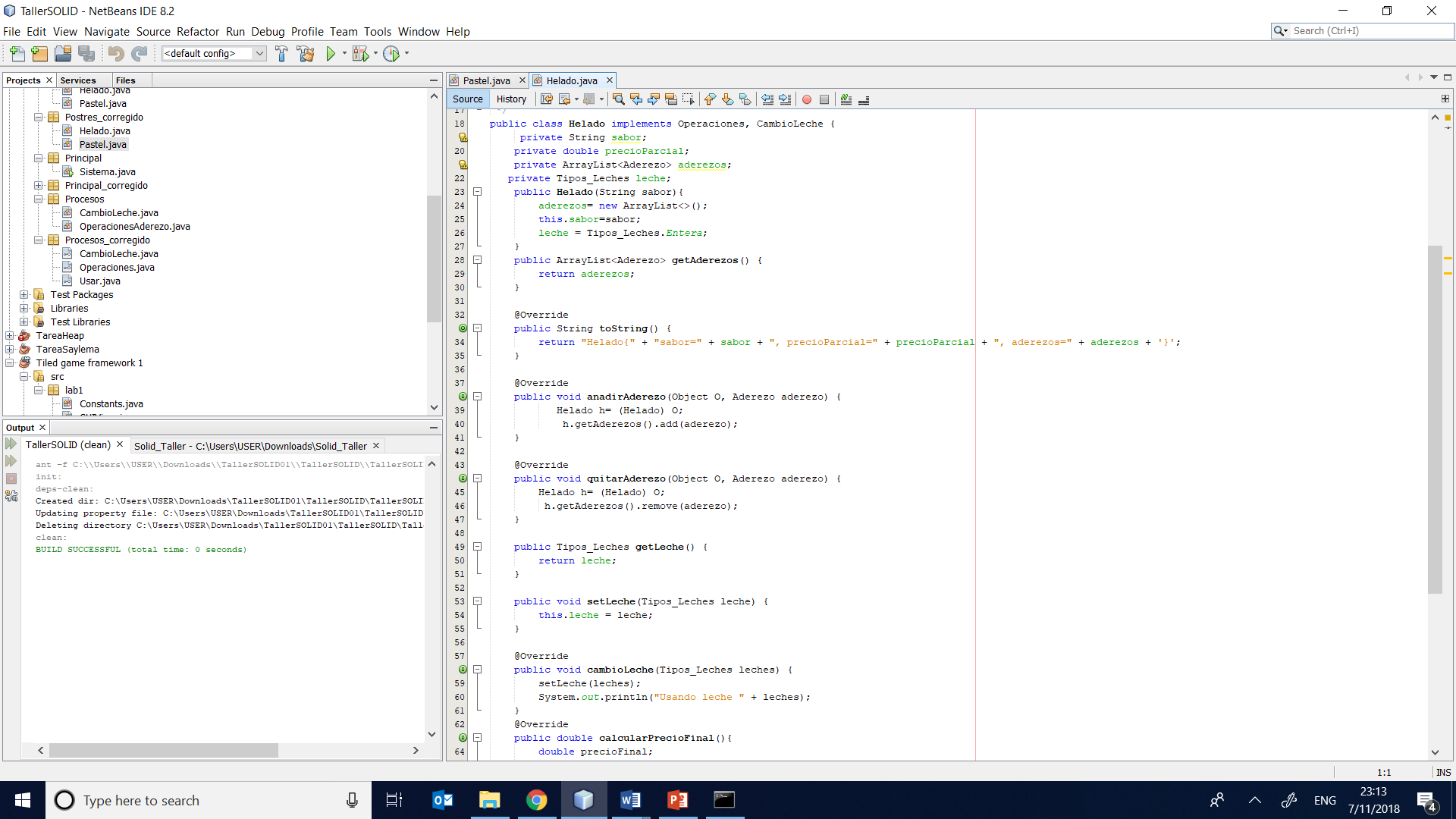
Samuel Braganza, Karina Saylema

* **Principio Single responsability y Open closed**

De acuerdo a nuestro análisis, métodos como calcularPrecioFinal(): double, agregarAderezo(O: Object, aderezo: Aderezo): void y quitarAderezo(O: Object, aderezo: Aderezo): void aparecían en ambas clases, Pastel y Helado . Los métodos del aderezo estaban en una interfaz llamada OperacionesAderezo declaradas e **implementadas** lo cual esta mal porque en una interfaz solo se los declara y en las clases que implementen esta interfaz sobreescribe métodos dependiendo de lo que se quiera hacer con ellos (Se hace la implementación). Y, como existía agregarAderezo para Pastel y Helado, entonces al momento de implementar esa interfaz se debía de sobrescribir ambos métodos, pero **un Helado no debe de implementar el método agregarAderezo ( Postre: p, Aderezo: a ): void, porque este se lo agregaría a un Postre.**

Código que **rompe** con el principio de Open closed y single responsability



 Codigo corregido

**Interface Operaciones**

