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 fuente para que cumpla con los principios SOLID.

# Antecedentes:

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, la misma fórmula se aplica para todos los productos, pero si se desea cambiar la formula, debe hacerse para todos los productos.

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, si el cliente lo prefiere, sin ningún costo adicional. Sin embargo, para los pasteles no se puede utilizar la leche deslactosada porque se daña la mezcla.

Se le encarga a su equipo que analice el código y haga que cumpla con los principios SOLID para obtener un código más limpio.

# Indicaciones

1. Crear un repositorio en Github con la versión base del proyecto.
2. Como grupo explique por qué o cómo se está violando cada principio SOLID. Agregar la explicación al archivo README.md del repositorio.
3. El taller es colaborativo, por lo tanto, todos deben subir cambios al repositorio remoto.

# SOLID

El sistema ejecuta una pequeña secuencia de pasos para probar el funcionamiento del código fuente:

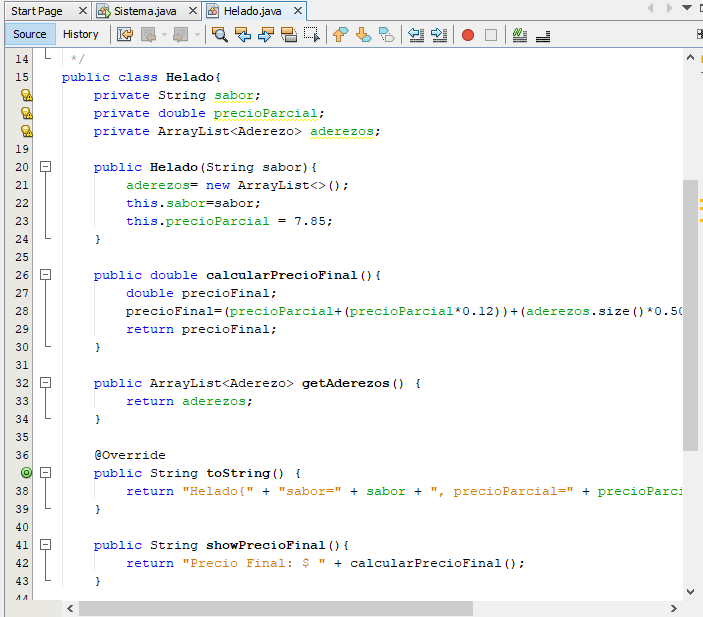
1. Producir un helado de vainilla y una torta de chocolate
2. A ambos agregarles CREMA y FRUTILLAS como aderezos
3. A ambos cambiar el tipo de leche por Leche Deslactosada
4. Finalmente mostrar el precio final de cada postre.

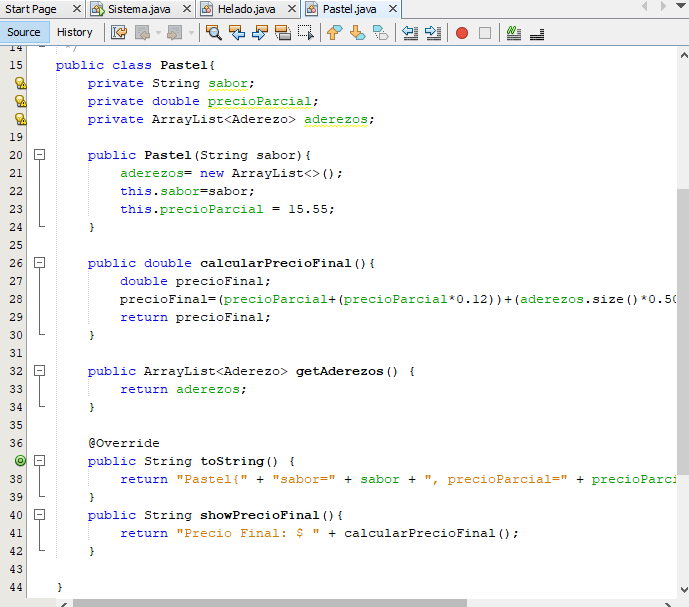
Analice las siguientes partes del código, explique su análisis en el archivo README.md y corrija el código para que cumpla con los principios SOLID.

1. Clases Helado y Pastel. Tienen mucha similitud, se debería crear una clase padre llamada Postre.

//**Principio de sustitución de Liskov**

Sí porque ambas clases, helado y pastel, tienen los mismos métodos, atributos y comportamiento entonces está bien crear una clase padre abstracta. Ya que en la clase padre se declararán los métodos y variables que podrán ser usados por las clases que extiendan de esta.

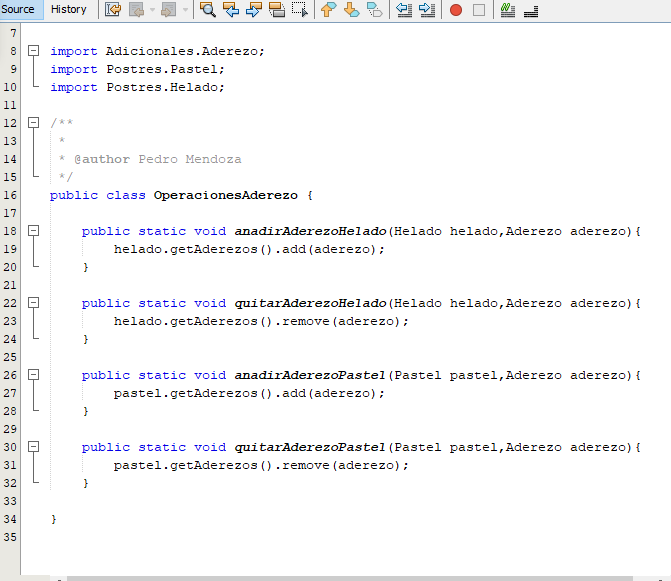




1. Clases Procesos.OperacionesAderezo y Postre. ¿Es necesaria la clase OperacionesAderezo?. Se puede incluir dentro de postre un método para agregar un aderezo y para quitar un aderezo.

//**Principio de sustitución de Liskov**

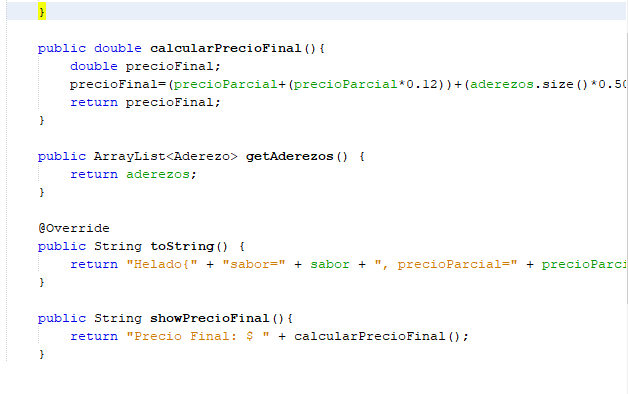
No es estrictamente necesaria la clase OperacionAderezo, se puede incluir dentro de Postre ya que no está sobrecargada. El hecho de que Postre se quede con todos los métodos que tenía OperacionesAderezo cumple el principio de sustitución de Liskov ya que solo esta clase se encargará de la modificación y administración de los postres almacenados en el sistema.



1. Métodos calcularPrecioFinal() y showPrecioFinal() están muy relacionados, deben estar en otra clase por si cambia la fórmula de cálculo. La clase nueva debe llamarse Procesos.ManejadorDePrecio.

//**Principio de responsabilidad única**

Al hacer la clase ManejadorDePrecio liberamos de métodos a la clase Postre y le damos la responsabilidad única a esta clase de encargarse del cálculo de los precios que tendrán los postres, haciendo así que la clase postres y sus subclases: helado y pastel, no tengan la responsabilidad de manejar el tema de los precios.

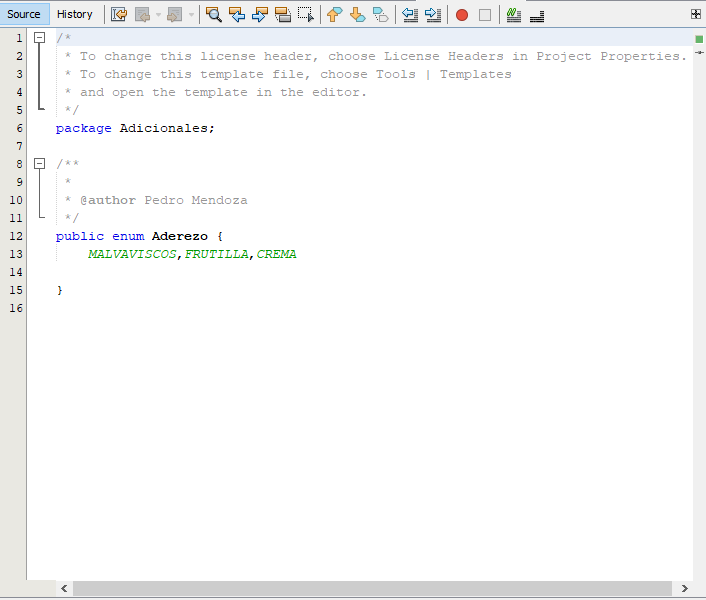


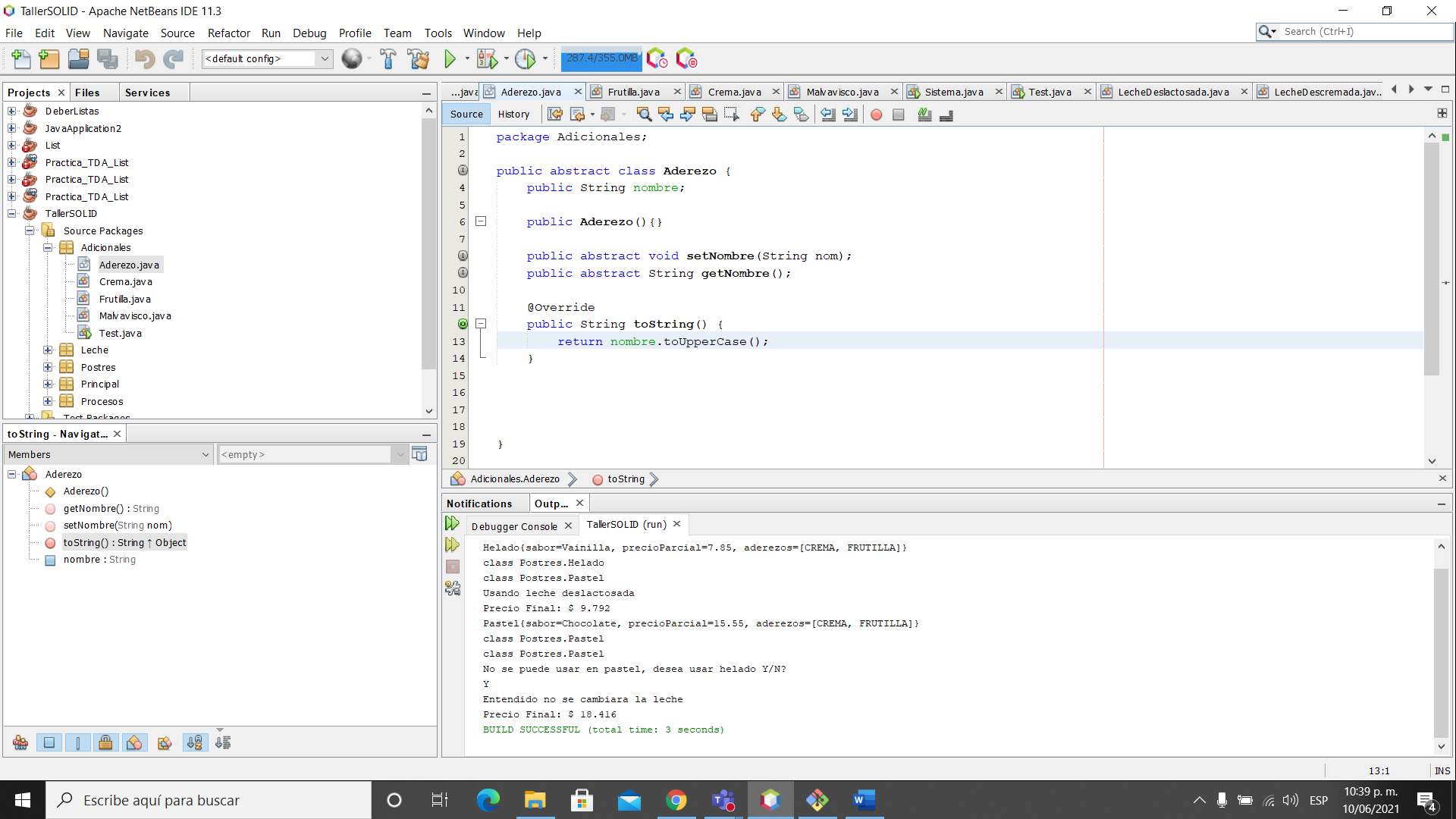
1. Enum Adicionales.Aderezo es muy estático, debe convertirse en clase abstract con un atributo nombre y un método abstracto setNombre para que cada tipo de aderezo sea una subclase de Aderezo e implemente dicho método. También, sobrescriba el método toString() en la clase Aderezo, para que devuelva el nombre del aderezo en mayúsculas.

//**Principio de inversión de dependencia**

Esto se debe a que los métodos de alto nivel no deberían depender de los módulos de bajo nivel como es el caso de los métodos dados, ya que según el principio de inversión de dependencia estos solo deberían depender de abstracciones.

Debido a esto se tuvo que modificar la clase aderezo y volverla una clase abstracta para que sus subclases, que son cada tipo de aderezo, cumplan este principio.

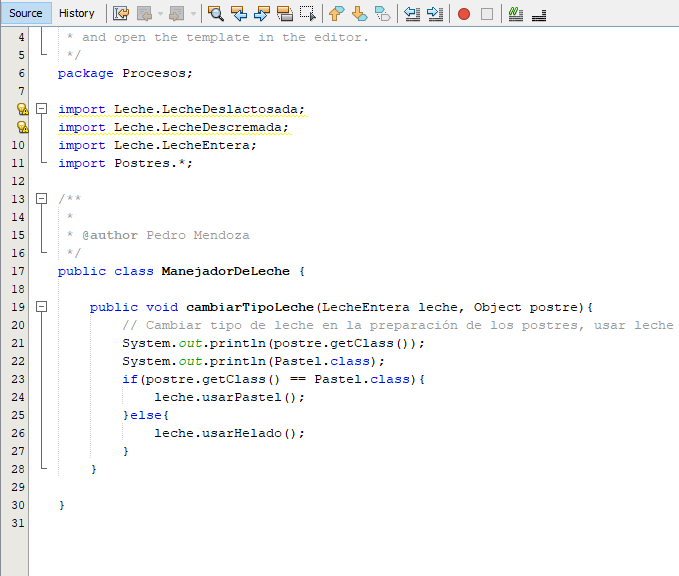


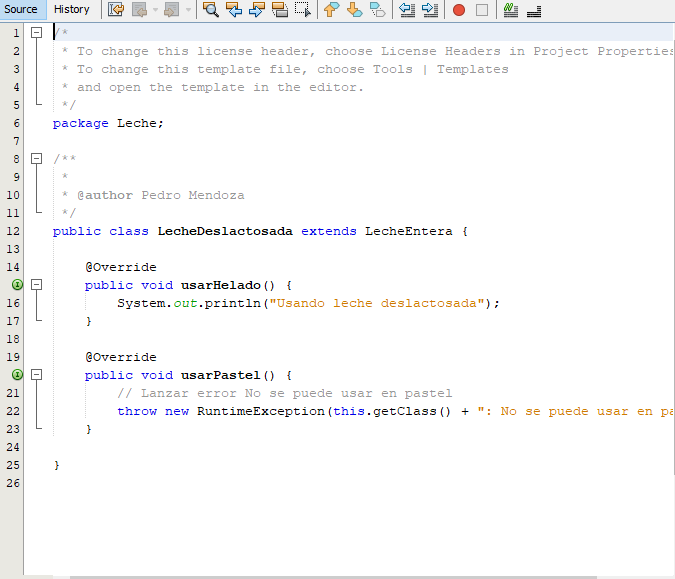


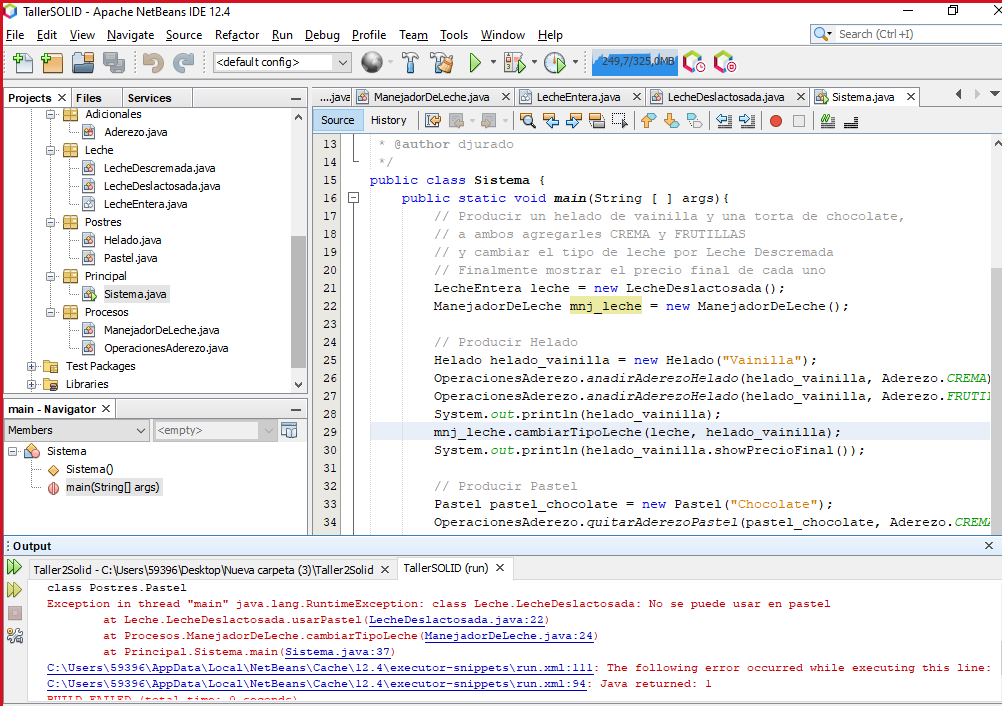
1. Paquete Leche y la clase Procesos.ManejadorDeLeche. En el main descomente las instrucciones para realizar el cambio del tipo de leche utilizada en cada postre, luego analice como solucionar el error generado en la clase Leche.LecheDeslactosada.

//**Principio de segregación de interfaces**

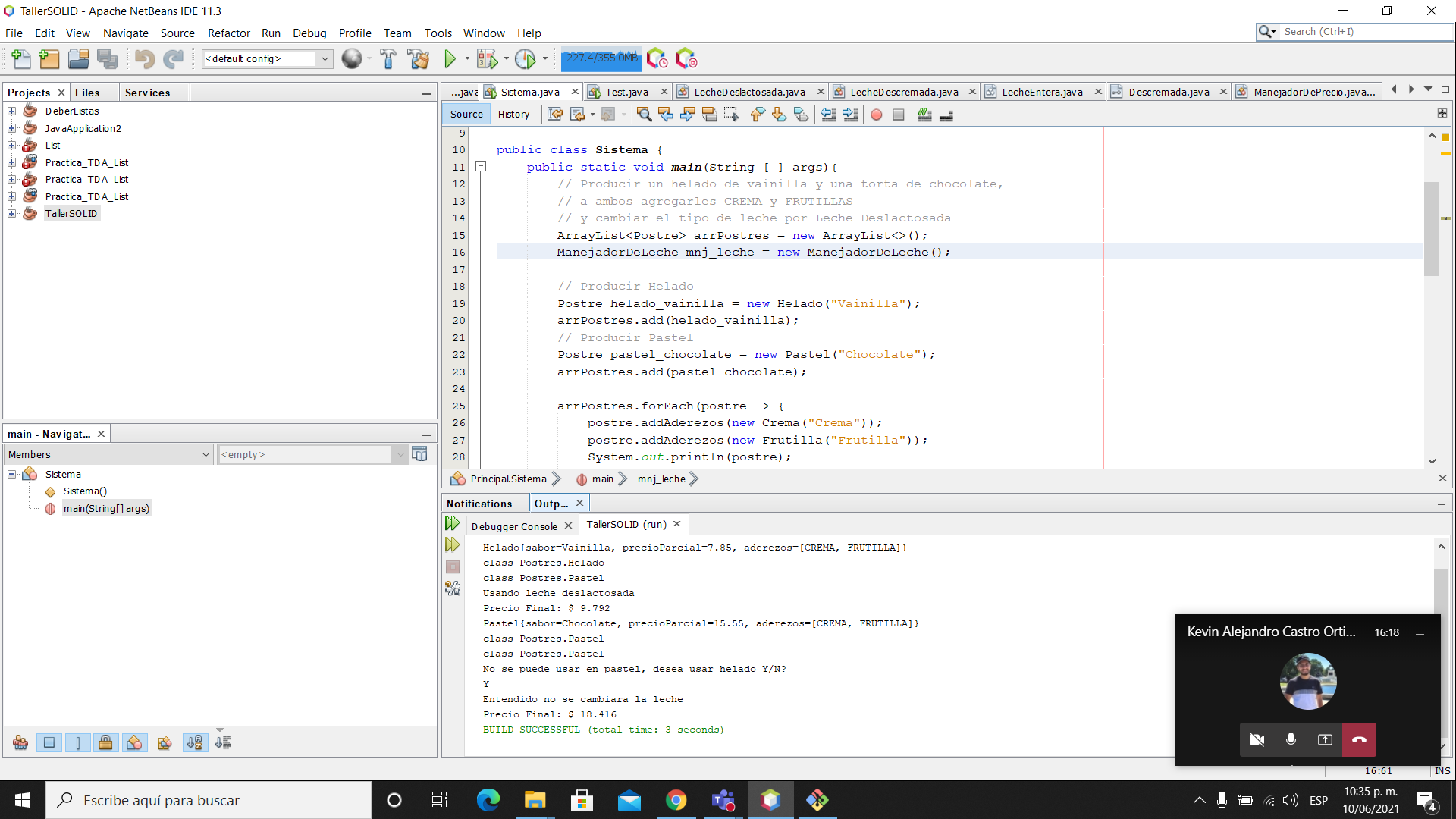
Ninguna clase debería depender de métodos que no usa, por eso se resuelve el conflicto generado creando distintas interfaces que provean métodos puntuales a las distintas subclases de la superclase lecheEntera, ya que de esta manera cada subclase podrá implementar los métodos que si les son de utilidad con el fin de no tener la necesidad de implementar métodos inútiles en esas clases.







1. Cambie el main por el siguiente código y **ajuste** según su propia implementación.



    public static void main(String [ ] args){

        // Producir un helado de vainilla y una torta de chocolate,

        // a ambos agregarles CREMA y FRUTILLAS

        // y cambiar el tipo de leche por Leche Deslactosada

        ArrayList<Postre> arrPostres = new ArrayList<>();

        ManejadorDeLeche mnj\_leche = new ManejadorDeLeche(new LecheDescremada());

        // Producir Helado

        Postre helado\_vainilla = new Helado("Vainilla");

        arrPostres.add(helado\_vainilla);

        // Producir Pastel

        Postre pastel\_chocolate = new Pastel("Chocolate");

        arrPostres.add(pastel\_chocolate);

        arrPostres.forEach(postre -> {

            postre.addAderezos(new Crema());

            postre.addAderezos(new Frutilla());

            System.out.println(postre);

            mnj\_leche.cambiarTipoLeche(postre);

            System.out.println(ManejadorDePrecio.showPrecioFinal(postre));

        });

    }