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.
2. 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.

3. 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.
4. 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.
5. 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.
6. 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));
        });

    }
```

RESPUESTAS

1.

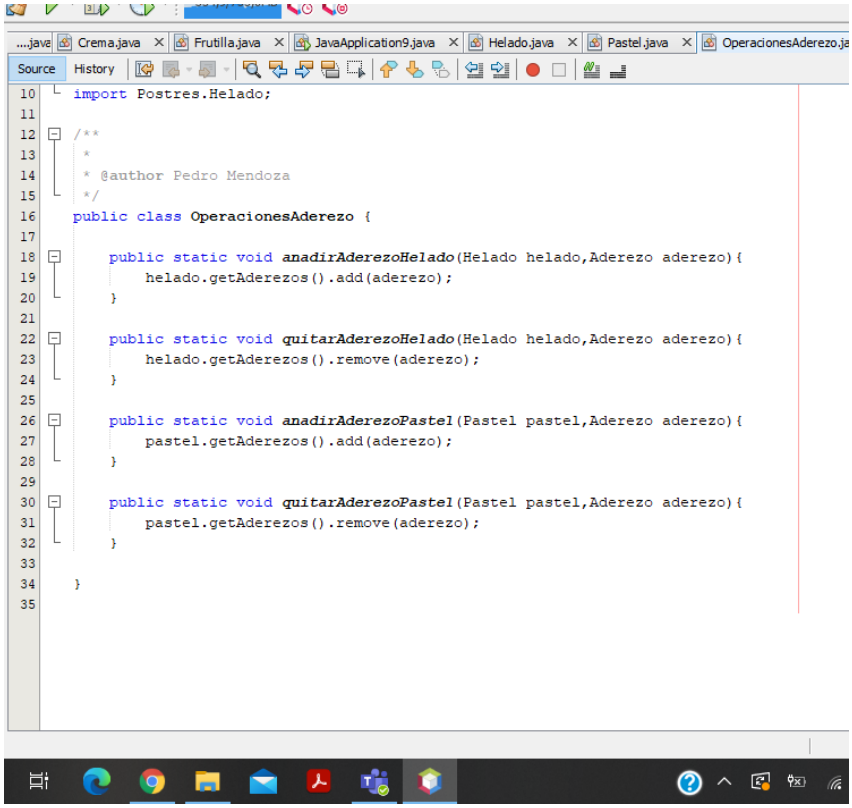
La clase Helado y la clase Pastel, no cumplen el principio de responsabilidad única, y en este caso dos clases tienen la misma responsabilidad en crear un postre, por tanto se puede delegar la responsabilidad de creación a una única superclase delas cuales hereden ambas.

```java
/**
 * @author Pedro Mendoza
 */
public class Helado{
    private String sabor;
    private double precioParcial;
    private ArrayList<Aderezo> aderezos;

    public Helado(String sabor){
        aderezos= new ArrayList<>();
        this.sabor=sabor;
        this.precioParcial = 7.85;
    }

    public double calcularPrecioFinal(){
        double precioFinal;
        precioFinal=(precioParcial+(precioParcial*0.12))+(aderezos.size()*0.50);
        return precioFinal;
    }

    public ArrayList<Aderezo> getAderezos() {
        return aderezos;
    }

    @Override
    public String toString() {
        return "Helado{" + "sabor=" + sabor + ", precioParcial=" + precioParcial + ", ad
    }

    public String showPrecioFinal(){
        return "Precio Final: $ " + calcularPrecioFinal();
    }
```

```java
*/
public class Pastel{
    private String sabor;
    private double precioParcial;
    private ArrayList<Aderezo> aderezos;

    public Pastel(String sabor){
        aderezos= new ArrayList<>();
        this.sabor=sabor;
        this.precioParcial = 15.55;
    }

    public double calcularPrecioFinal(){
        double precioFinal;
        precioFinal=(precioParcial+(precioParcial*0.12))+(aderezos.size()*0.50);
        return precioFinal;
    }

    public ArrayList<Aderezo> getAderezos() {
        return aderezos;
    }

    @Override
    public String toString() {
        return "Pastel{" + "sabor=" + sabor + ", precioParcial=" + precioParcial + ", aderezos=" + aderez
    }
    public String showPrecioFinal(){
        return "Precio Final: $ " + calcularPrecioFinal();
    }

}
```

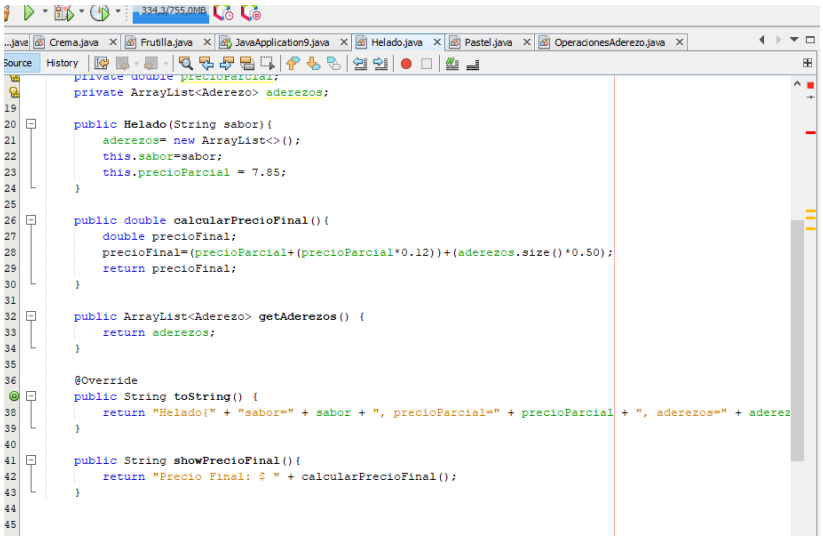
2.-

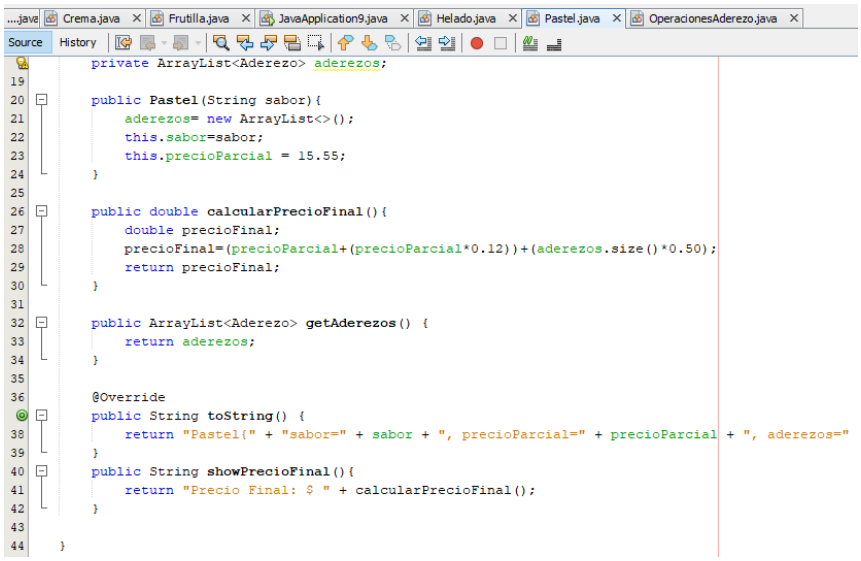
No se está cumpliendo el principio de OPEN-CLOSED ya que en el literal 1 se eliminó la clase Helado y Pastel y se creó la clase postre, lo que se debe hacer es eliminar la clase operaciones aderezo ya que esta contiene métodos específicos para pastel y helado, y agregarla a la clase postre, si no eliminamos operaciones aderezo en el caso que nosotros vayamos a agregar otro postre el programa no va a funcionar y se estaría violando OPEN-CLOSED.



3.-

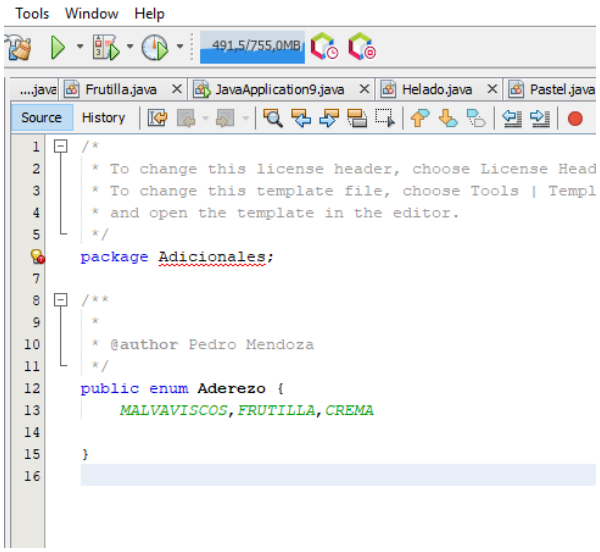
Inclumple el Single Responsability Principle, el cual dictamina que una clase debería tener solo una responsabilidad, en este caso el calcular precios es una responsabilidad que la cumplen dos clases por igual, por lo que se podría separar en una clase aparte para que no haya repetición de código.





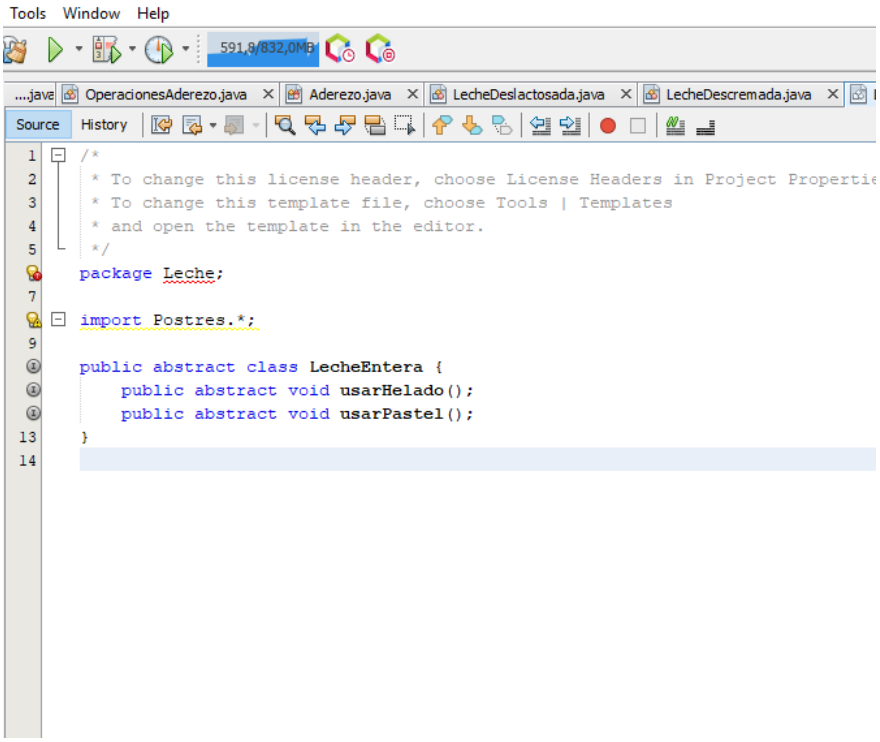
4.-

Está incumpliendo el Dependency Inversión Principle ya que esta clase dependía de objetos concretos como malvaviscos, frutilla y crema, pero que pasa si en un momento nosotros queremos agregar dos frutillas como aderezo, el programa no iba a permitir eso, la solución es hacer que dependan de instancias y así cualquier aderezo que se le ponga al postre se va a agregar.



5.

La solución más factible para el problema del error es agregarla al manejador, como una excepción esperada y manejada ya que se nos informa de ante mano que no se puede usar ese tipo de leche para el pastel, y para el control de leche, usar un constructor que determine el tipo de leche que se usara en la producción.



6.- Hay que adaptar el main a todos los cambios realizados. Finalmente obtenemos el siguiente main.