LINK DEL REPOSITORIO:

https://github.com/farincon114/principios-solid.git

Estos principios se llamaron S.O.L.I.D. por sus siglas en inglés:

S: Single responsibility principle o Principio de responsabilidad única

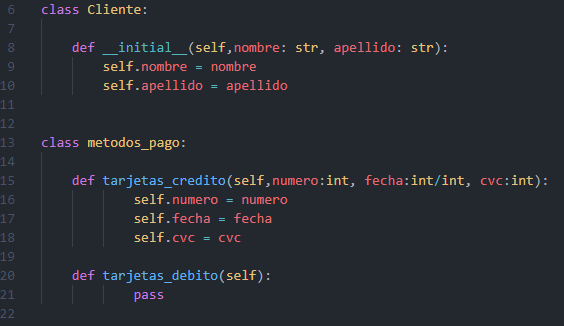
O: Open/closed principle o Principio de abierto/cerrado

L: Liskov substitution principle o Principio de sustitución de Liskov

I: Interface segregation principle o Principio de segregación de la interfaz

D: Dependency inversion principle o Principio de inversión de dependencia

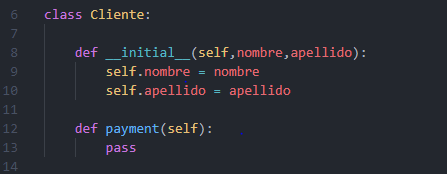
**S: Principio de responsabilidad única**



INCORRECTO

Se estan agregando dos responsabilidades en este ejemplo, la de obtener informacion

del cliente y la de asignar un metodo de pago

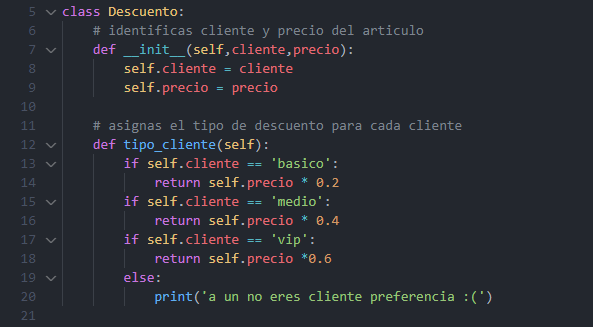


CORRECTO

en este se puede evidenciar que se corrige el error ya que asignamos una clase

para obtener la informacion del cliente y otra para asignar un metodo de pago

**O: Open/closed principle o Principio de abierto/cerrado**



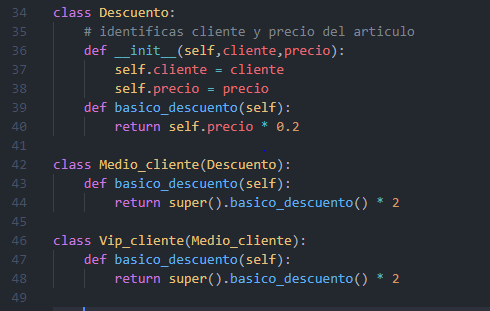
INCORRECTO

esta mal por que estamos modificando el valor de la funcion del descuento todo

el tiempo, recuerden que, debe estar abierta para su extencion , pero cerrada

para su modificacion, en este caso debemos extender la funcion pero cerrado para su

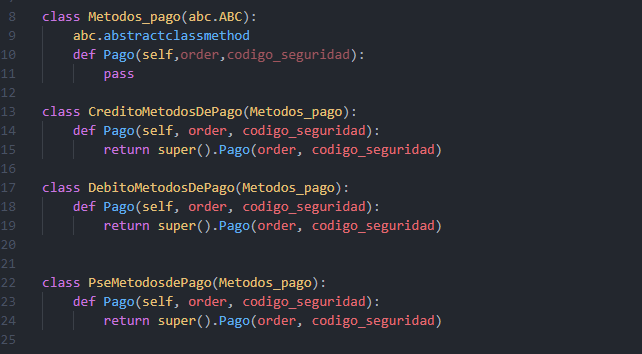
modificacion



CORRECTO

como pudimos observar en este código por medio de clases extendimos las funciones (descuento), pero no modificamos ninguna cerrada

**L: Liskov substitution principle o Principio de sustitución de Liskov**



INCORRECTO:

se crea un metodo con orden de pago y un codigo de seguridad para las opciones

de pago pero encontramos que el pago pse no funciona con codigo de seguirdad si no

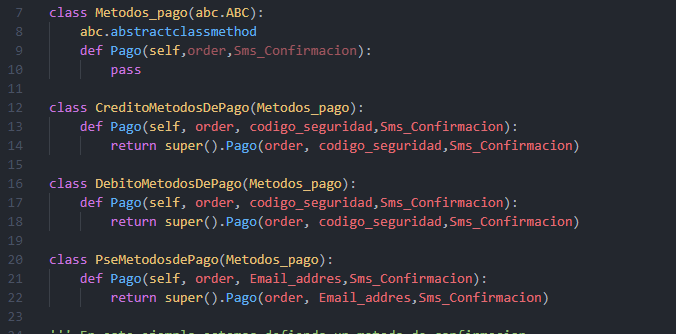
con correo electronico estamos faltando al principio de sustitucion de liskov



CORRECTO:

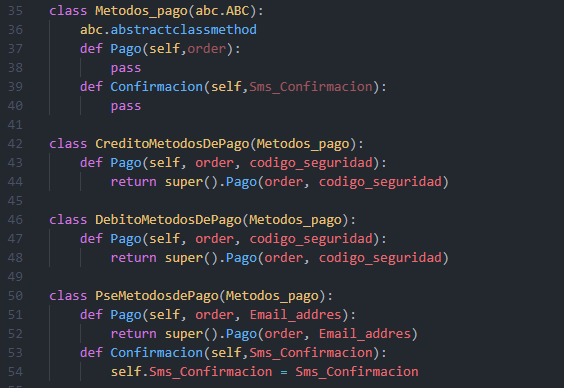
ahora lo que se realizo fue crear una orden de pago en la superclase y en cada subclase se creo un metodo para definir como se pagara la orden de pago.

**I: Interface segregation principle o Principio de segregación de la interfaz**



INCORRECTO:

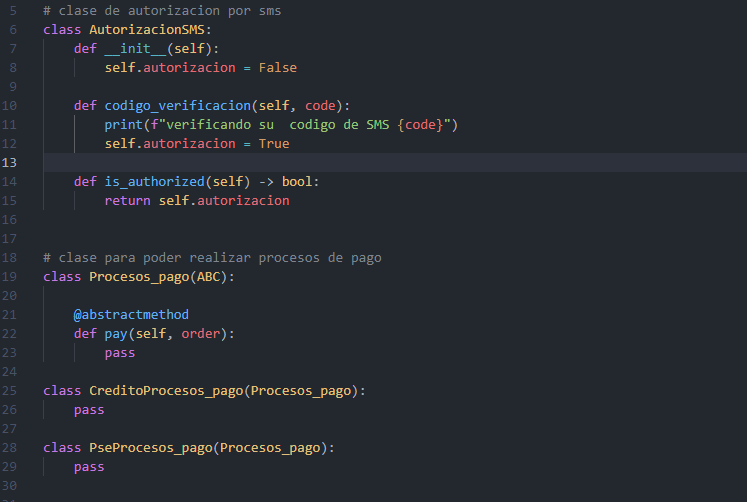
En este ejemplo estamos defiendo un metodo de confirmación de SMS de pago a todas class de metodos de pago, pero en realidad la unica que lo va nesecitar el el pago por PSE ya que los demás tienen un codigo de seguridad en el cual puede pasar el pago estariamos faltando al principio de segregacion de interfaces, la mejor forma de crearlo seria el siguiente ejemplo



**CORRECTO:**

creamos una interfaz que va tener como opcion la confirmación de pago por el sms pero solo al que realmente lo va a nesecitar, sin obligar a los demas metodos a usarla asi cumplimos con el principio

**D: Dependency inversion principle o Principio de inversión de dependencia**



INCORRECTO:

en este ejemplo creamos un funcion para autorizacion de pago, pero que ocurre si en la autorización solo la tenemos con sms, y queremos agregar mas formas, nos tocaria modificar todo el codigo ya que dependemos un clase estariamos faltando al principio en ese caso seria mejor realizarlo como el otro ejemplo



CORRECTO:

en este caso realizamos creamos una abstraccion de autorizacion para que se pueda realizar por medio de metodos abstractos mas no que dependan