**Escuela Superior Politécnica del Litoral**

**Diseño de Software**

**Taller 03**

**Principios SOLID**

**Integrantes del equipo**


- Estrada Santana Michael Bryan
- Pasaca Paladines Noelia Alexandra
- Plúas Muñoz Juan Pablo
- Sánchez Villacreses Samantha Sharid


**Dr. Carlos Mera**

**Guayaquil, 27 de octubre del 2022**

Tabla de contenido

# Sección A

Elabore, extienda o adapte un ejemplo ilustrativo para cada principio de diseño SOLID. Puede utilizar diagramas UML o código fuente en Java.

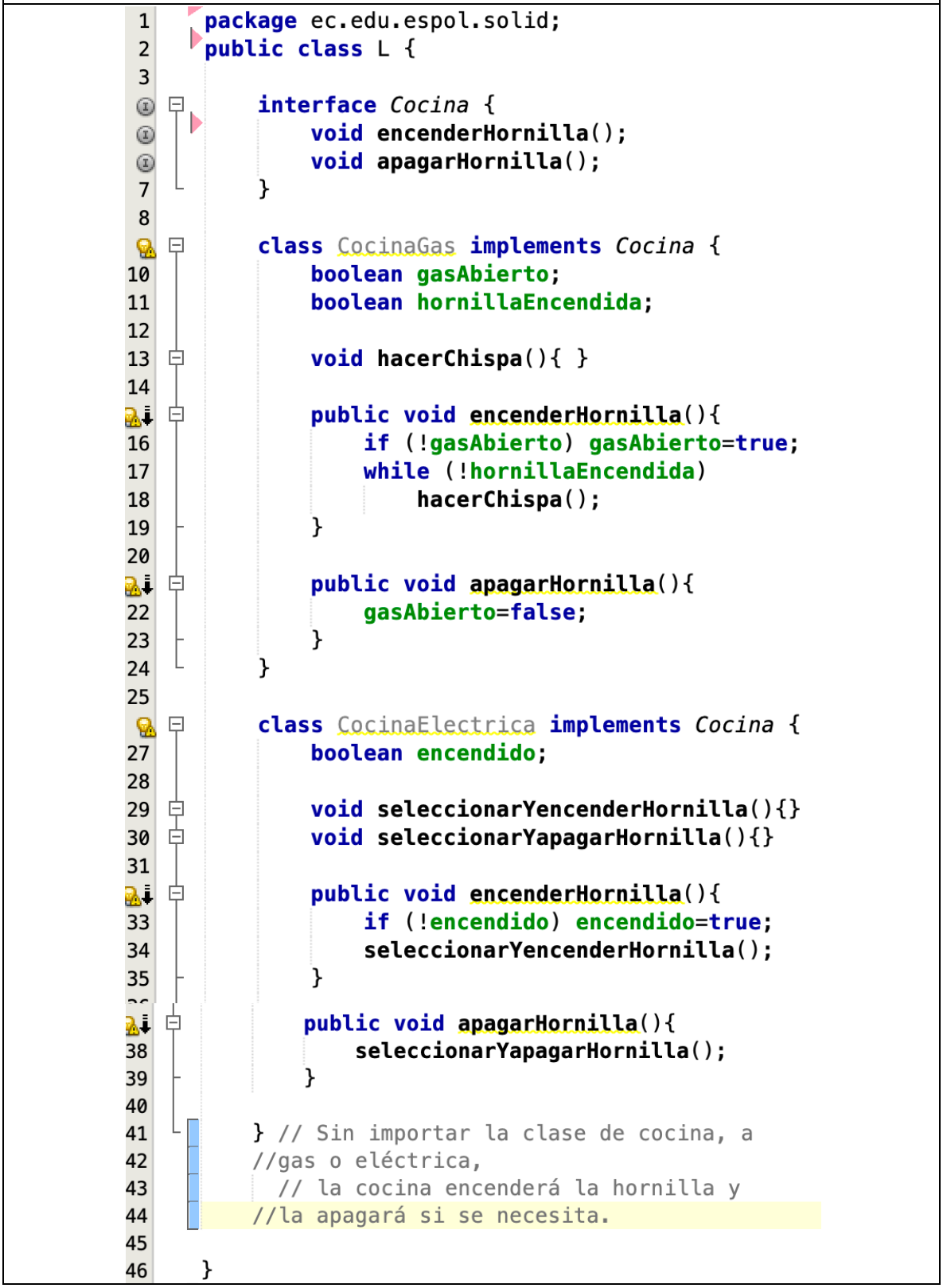
## Ilustración de SRP

```java
package ec.edu.espol.solid;

public class S {

    /* CODIGO QUE VIOLA EL PRINCIPIO

    class ArtículoRevista {

    private String nombreArtículo;
    private String nombreRevista;
    private String seccionRevista;
    private String[] autores;
    private String[] fuentes;

    void agregarImagen(){ }
    void eliminarImagen(){ }
    void editarTexto(){ }
    void editarFondo(){ }

    // constructores, getters y setters

    }

    */



    class ArtículoRevista { //La clase solo almacena la información del artículo

        private String nombreArtículo;
        private String nombreRevista;
        private String seccionRevista;
        private String[] autores;
        private String[] fuentes;

        // constructores, getters y setters

    }

    class SoftwareDeEdicion { //Se crea una clase que SOLO se encargue de
        //la edición del articulo

        void agregarImagen(){ }
        void eliminarImagen(){ }
        void editarTexto(){ }
        void editarFondo(){ }
    }



}
```

Ilustración de OCP

```java
package ec.edu.espol.solid;

/**
 *
 * @author sam sung
 */
public class O {

    interface enjoyable {

        void fun();
    }

    class Ride implements enjoyable {

        @Override
        public void fun() {
            System.out.println("I love car rides!");
        }
    }

    class Walk implements enjoyable {

        @Override
        public void fun() {
            System.out.println("Walking is my favorite hobby!");
        }
    }
    //Se puede seguir añadiendo más clases que implementen la misma
    //interfaz,
    //pero queda cerrado a modificaciones posteriores tanto a cada
    //clase como a la interfaz.
}
```

Ilustración de LSP



```
1  package ec.edu.espol.solid;
2  public class L {
3
4      interface Cocina {
5          void encenderHornilla();
6          void apagarHornilla();
7      }
8
9      class CocinaGas implements Cocina {
10         boolean gasAbierto;
11         boolean hornillaEncendida;
12
13         void hacerChispa(){ }
14
15         public void encenderHornilla(){
16             if (!gasAbierto) gasAbierto=true;
17             while (!hornillaEncendida)
18                 hacerChispa();
19         }
20
21         public void apagarHornilla(){
22             gasAbierto=false;
23         }
24     }
25
26     class CocinaElectrica implements Cocina {
27         boolean encendido;
28
29         void seleccionarYencenderHornilla(){}
30         void seleccionarYapagarHornilla(){}
31
32         public void encenderHornilla(){
33             if (!encendido) encendido=true;
34             seleccionarYencenderHornilla();
35         }
36
37         public void apagarHornilla(){
38             seleccionarYapagarHornilla();
39         }
40
41     } // Sin importar la clase de cocina, a
42     //gas o eléctrica,
43       // la cocina encenderá la hornilla y
44     //la apagará si se necesita.
45
46 }
```

Ilustración de ISP

```java
package ec.edu.espol.solid;

/**
 *
 * @author sam sung
 */
public class I {

// se realiza un sistema educativo donde al iniciar sesion se muestra
    //la informacion de los usuarios

    /**
     * CODIGO QUE VIOLA EL PRINCIPIO
    class SistemaAcademico{
        void mostrarInformacion(){
            //codigo que muestra la informacion al iniciar sesion
        };
    };

    class Estudiante extends SistemaAcademico{
        @Override
        void mostrarInformacion(){
            //codigo que muestra la informacion al iniciar sesion
        };
    };
    class Maestro extends SistemaAcademico{
        @Override
        void mostrarInformacion(){
            //codigo que muestra la informacion al iniciar sesion
        };
    };
    */

     interface SistemaAcademicoEstudiantil{
         //...
         void mostrarInformacionEstudiante();
         //...
     };
     interface SistemaAcademicoProfesor{
         //...
         void mostrarInformacionProfesor();
         //...
     };

     class Estudiante implements SistemaAcademicoEstudiantil{
         //...
         @Override
         public void mostrarInformacionEstudiante(){};
         //...
     };
     class Maestro implements SistemaAcademicoProfesor{
         //...
         @Override
         public void mostrarInformacionProfesor(){};
         //...
     };
}
```

# Ilustración de DIP

```java
package ec.edu.espol.solid;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Optional;

public class D {

    public interface CustomerDao {

        Optional<Customer> findById(int id);

        List<Customer> findAll();

    }

    public class Customer {

        private final String name;

        public Customer(String name) {
            this.name = name;
        }

        public String getName() {
            return name;
        }

        @Override
        public String toString() {
            return "Customer{" + "name=" + name + '}';
        }
    }


    public class SimpleCustomerDao implements CustomerDao {

        private Map<Integer, Customer> customers = new HashMap<>();

        public SimpleCustomerDao(Map<Integer, Customer> customers) {
            this.customers = customers;
        }

        @Override
        public Optional<Customer> findById(int id) {
            return Optional.ofNullable(customers.get(id));
        }

        @Override
        public List<Customer> findAll() {
            return new ArrayList<>(customers.values());
        }
    }

    public class CustomerService {

        private final CustomerDao customerDao;

        public CustomerService(CustomerDao customerDao) {
            this.customerDao = customerDao;
        }

        public Optional<Customer> findById(int id) {
            return customerDao.findById(id);
        }

        public List<Customer> findAll() {
            return customerDao.findAll();
        }
    }
```

Referencia: https://github.com/eugenp/tutorials/tree/master/patterns-modules/dip/src/main/java/com/baeldung/dip

# Sección B

Dado el siguiente código con cuatro clases, considere el caso en que se desea agregar una nueva forma de notificación (ej. Signal). Identifique los principios SOLID que se están violando. Para cada principio, explique la razón y corrija el código de tal forma que ya no se lo viole. Si lo considera necesario, usted puede crear interfaces, clases o nombres de métodos. Incluso puede utilizar diagramas de clase.

## Clase Compra

Principio que se viola: Open-Closed Principle

Explicación: ya que es la misma estructura de código del método comprar.

Solución: la solución óptima es que tenga como parámetro una interfaz que esté implementada tanto en Pago como en PagoPayPal.

Clase Compra.java

```
1  package seccionB;
2
3  import java.util.List;
4
5  public class Compra {
6      private Pago pago;
7      private PagoPayPal pagoPayPal;
8      private List articulos;
9
10
11     /**
12     public Compra(Pago pago){
13         //constructor
14     }
15     public Compra(PagoPayPal pagoPayPal){
16         //constructor
17     }
18     */
19
20     public Compra(Pagar pagar){
21         //constructor
22     }
23     public void agregarArticulo(Articulo articulo){
24         //agg articulo a la compra
25     }
26     public void removerArticulo(Articulo articulo){
27         //rmv articulo a la compra
28     }
29 }
```

Interfaz Pagar.java

```java

package seccionB;


public interface Pagar {
    void realizarCobro(double monto);
}
```

## Clase Pago y Clase Pago PayPal

Principio que se viola: Liskov Substitution Principle y Simple Responsibility Principle

Explicación: ya que tanto Pago y PagoPayPal comparten la misma función, además, la clase pago tiene varias responsabilidades.

Solución: la mejor solución es crear una interfaz y que esta interfaz con esta misma función implemente a estas clases, y también que se ejecute sin ninguna variación en las dos clases. Además, crear las clases Impuesto y Factura que se encarguen de realizar sus responsabilidades correspondientes.

Clase Pago.java

```java

package seccionB;

public class Pago implements Pagar{




    /**
     * public void realizarCobro(double monto){ //cargar el monto de compra al
     * medio de pago }
     * public void calcularImpuestoFactura(){ //calcula iva }
     * public void generarFactura(){ //genera nueva factura }
     */

    @Override
    public void realizarCobro(double monto) {
        //cargar el monto de compra al  medio de pago
    }
}

```

Clase Factura.java

```java

package seccionB;


public class Factura {
    //atrb
    public void generarFactura(Pago pago, Impuesto imp){
        //genera nueva factura
    }
}

```

Clase Impuesto.java

```java

package seccionB;

//
public class Impuesto {
    //atrb
    public void calcularImpuestoFactura(){
        //calcula iva
    }
}

```

Clase PagoPayPal.java

```java
package seccionB;

public class PagoPayPal implements Pagar{
    private boolean loggedIn;//validacion de conexion a cuenta paypal

    @Override
    public void realizarCobro(double monto){
        if(!loggedIn){
            /*
            TurnLogAccount();
            /*something*/
        }
        //cargar el monto de compra al medio de pago
    }
}
```

## Clase Notificación

Principio que se viola: Open-Closed Principle

Explicación: Se debe modificar el método cada vez que queremos añadir una nueva forma de notificación.

Solución: Crear una interfaz Notificación que tenga un método que se pueda implementar en las otras clases con las otras formas de notificación, de manera que se pueda extender el código y no se tenga que modificar el método.

## Interface Notificación

```java

package seccionB;



public interface Notificacion{
    public void notificar(Pagar pago);
}

class NotiEmail implements Notificacion{
   public void notificar(Pagar pago){
       //enviaremail
   }
}

class NotiSMS implements Notificacion{
    public void notificar (Pagar pago){
        //enviarSMS
    }
}
```