Integrantes:

Katherine Alexandra Tumbaco Sellan 202104485

Cesar Alejandro Arana Castro 202101663

Melisa Nathalia Ayllon Gutierrez 202109641

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.

Links ejemplo: https://github.com/romulets/JavaSOLID.git

Adaptación-Extensión

Single-Responsability Principle(SRP)

Problema

```java
package Problema;
import java.io.FileWriter;
import java.io.IOException;
public class Estudiante {
    private String correo;
    private int materias;
    private String documento;

    public void registrar() {
        if(!correo.contains("@")) {
            throw new RuntimeException("El correo debe contener el '@'");
        }
        if(materias<=0 || materias>15)
            throw new RuntimeException("El estudiante no es regular");
        FileWriter file;

    try {
        file = new FileWriter("database.txt");
        file.write(this.toString());
        file.close();
    } catch (IOException e) {
        throw new RuntimeException("No se puede guardar el estudiante", e);
    }
}
    @Override
    public String toString() {
        return String.format("%s -|- %s ", correo, documento);
    }

    public void setCorreo(String correo) {
        this.correo=correo;
    }
}
```

```java
package Problema;

public class SRPProblema {
     public static void main(String[] args) {
           Estudiante estudiante = new Estudiante() {{
                setCorreo("prueba@institucion.edu.pais");
            }};

            try {
                estudiante.registrar();
                System.out.println("Estudiante registrado correctamente");

            } catch (RuntimeException e) {
                e.printStackTrace();
            }
        }
}
```

Solución

```java
package Solucion;

public class ValidarEmail {
    public static void validar(String correo) {
    if(!correo.contains("@")) {
        throw new RuntimeException("El correo debe contener el '@'");
    }
    }
}
```

```java
package Solucion;

public class ValidadorMaterias {
    public static void validar(int materias) {
        if ( materias<=0 || materias>15)
            throw new RuntimeException("El estudiante no es regular");
    }

}
```

```java
package Solucion;

import java.io.FileWriter;
import java.io.IOException;

public class RepositorioEstudiante {
    public void registro(Estudiante estudiante) {
        FileWriter file;

        try {
            file = new FileWriter("database.txt");
            file.write(estudiante.toString());
            file.close();
        } catch (IOException e) {
            throw new RuntimeException("No se puede guardar el estudiante", e);
        }
    }


}
```

```java
package Solucion;

public class ServicioEstudiante {
    private RepositorioEstudiante repos;

    public ServicioEstudiante() {
        repos = new RepositorioEstudiante();
    }

    public boolean save(Estudiante estudiante) {
        try {

            estudiante.validar();
            repos.registro(estudiante);
            return true;

        } catch (RuntimeException e) {
            e.printStackTrace();
            return false;
        }
    }

}
```

```java
package Solucion;
public class Estudiante {
    private String correo;
    private int materias;
    private String documento;

    public void validar() {
        ValidarEmail.validar(correo);
        ValidadorMaterias.validar(materias);
}
    @Override
    public String toString() {
        return String.format("%s -|- %s ", correo, documento);
    }

    public void setCorreo(String correo) {
        this.correo=correo;
    }
}
```

```java
package Solucion;

import Problema.Estudiante;

public class SRPSolucion {
    public static void main(String[] args) {
            Estudiante estudiante = new Estudiante() {{
                setCorreo("prueba@institucion.edu.pais");
            }};

            try {
                estudiante.registrar();
                System.out.println("Estudiante registrado correctamente");

            } catch (RuntimeException e) {
                e.printStackTrace();
            }
        }
}
```

Open-Close Principle (OCP)

Problema

```java
package Problema;

public class ServicioTransporte {
    public void viaje(FormaDeTransporte transporte) {
        switch(transporte.getTransViaje()) {
        case CARRO:
            System.out.println("El pago por viajar en auto es: "+transporte.getValor());
            break;
        case AVION:
            System.out.println("El pago por viajar en avión es: "+transporte.getValor());
            break;

        }

    }
}
```

```java
package Problema;

public class FormaDeTransporte {
    private Transporte transViaje;
    private double valor;

    public FormaDeTransporte(Transporte transViaje, double valor) {
        setTransporte(transViaje);
        setValor(valor);

    }

    public Transporte getTransViaje() {
        return transViaje;
    }

    public void setTransporte(Transporte transViaje) {
        this.transViaje = transViaje;
    }

    public double getValor() {
        return valor;
    }

    public void setValor(double valor) {
        this.valor = valor;
    }
}
```

```java
package Problema;

public class OCPProblema {
    public static void main(String arg[]) {
        ServicioTransporte servicio = new ServicioTransporte();

        servicio.viaje(new FormaDeTransporte(Transporte.CARRO, 35.85));
        servicio.viaje(new FormaDeTransporte(Transporte.AVION, 385.67));

    }
}
```

```java
package Problema;

public enum Transporte {
    CARRO,
    AVION


}
```

Solución

```java
package Solucion;

public class FormaDeTransporte {
    private double valor;

    public FormaDeTransporte(double valor) {
        setValor(valor);

    }

    public double getValor() {
        return valor;
    }

    public void setValor(double valor) {
        this.valor = valor;
    }
}

```

```java
package Solucion;

public interface Transporte {
    void viaje(double valor);


}

```

```java
package Solucion;

public class TransporteAvion implements Transporte {
    @Override
    public void viaje(double valor) {
        System.out.println("El valor a pagar por usar un avión es: "+valor);

    }
}

```

```java
package Solucion;

public class TransporteCarro implements Transporte {
    @Override
    public void viaje(double valor) {
        System.out.println("El valor a pagar por usar un carro es: "+valor);

    }
}
```

```java
package Solucion;


public class ServicioTransporte {
    public void viaje(FormaDeTransporte formaTrans, Transporte  transporte) {
        transporte.viaje(formaTrans.getValor());



    }

}

```

```java
package Solucion;

public class OCPSolucion {
    public static void main(String arg[]) {
        ServicioTransporte servicio = new ServicioTransporte();

        Transporte carro = new TransporteCarro();
        Transporte avion = new TransporteAvion();

        servicio.viaje(new FormaDeTransporte(35.85),carro);
        servicio.viaje(new FormaDeTransporte(385.67),avion);

    }
}

```

Liskov Substitution Principle (LSP)

Problema

```java
package Problema;
public class CuentaComun{

    private double balance;

    public CuentaComun(double balance) {
        this.balance = balance;
    }

    public double getBalance() {
        return balance;
    }

    protected void setBalance(double balance) {
        this.balance = balance;
    }

    public void intereses() {
        //
    }

    public void comisionesPorTarjeta() {
        //
    }
}
```

```java
package Problema;
public class CuentaEstudiante extends CuentaComun {

    public CuentaEstudiante(double balance) {
        super(balance);
    }

    @Override
    public void intereses() {
        //
    }

    public void comisionesPorTarjeta() {
        //
    }
}
```

```java
package Problema;

import java.util.List;

public class CuentaServicio {
     public void interesesDiarios(List<CuentaComun> cuentas) {
          for (CuentaComun cuenta : cuentas) {
                cuenta.intereses();
                cuenta.comisionesPorTarjeta();
            }
     }
}
```

```java
package Problema;
public class CuentaSinIntereses extends CuentaComun {

    public CuentaSinIntereses(double balance) {
        super(balance);
    }

    @Override
    public void intereses() {
        // No permite aplicar intereses diarios
    }
}
```

```java
package Problema;
import java.util.ArrayList;
public class LSPProblema {
    public static void main(String[] args) {
        List<CuentaComun> cuenta = new ArrayList<>();
        CuentaServicio servicio = new CuentaServicio();

        cuenta.add(new CuentaComun(6000d));
        cuenta.add(new CuentaEstudiante(20000d));

        servicio.interesesDiarios(cuenta);
    }
}
```

Solución

```java
package Solucion;

public class CuentaComun {
    private double balance;

    public CuentaComun(double balance) {
        this.balance = balance;
    }

    public double getBalance() {
        return balance;
    }

    protected void setBalance(double balance) {
        this.balance = balance;
    }

    public void intereses() {
        //
    }

    public void comisionesPorTarjeta() {
        //
    }
}
```

```java
package Solucion;

import Solucion.CuentaComun;

public class CuentaEstudiante extends CuentaComun {
    private double limite;

    public CuentaEstudiante(double saldoInicial, double limite) {
        super(saldoInicial);
        this.limite = limite;
    }

    public double getLimite() {
        return limite;
    }

    public void setLimite(double limite) {
        this.limite = limite;
    }
}
```

```java
package Solucion;

import java.util.List;

public class CuentaServicio {
     public void interesesDiarios(List<Solucion.CuentaComun> cuenta
         for (Solucion.CuentaComun  cuenta : cuentass) {
                cuenta.intereses();
                cuenta.comisionesPorTarjeta();
           }
     }


}
```

```java
package Solucion;

import Solucion.CuentaComun;

public class CuentaSinIntereses extends CuentaComun {

    public CuentaSinIntereses(double balance) {
        super(balance);
    }

    @Override
    public void intereses() {
        // No permite aplicar intereses diarios
    }
}
```

```java
package Solucion;

import java.util.ArrayList;

public class LSPSolucion {
    public static void main(String[] args) {
        List<CuentaComun> cuenta = new ArrayList<CuentaComun>();
        CuentaServicio servicio = new CuentaServicio();

        cuenta.add(new CuentaComun(6000d));
        cuenta.add(new CuentaEstudiante(20000d,5000000d));

        servicio.interesesDiarios(cuenta);
    }
}
```

Interface Segregation Principle (ISP)

Problema

```java
package Problema;

public interface Empleado {
    public void tiempoextra();
    public void salario();
    public void fondosdereserva();
}
```

```java
package Problema;

public class Empleados implements Empleado{
    private String nombre;
    private double salario;

    public Empleados(String nombre, double salario) {
        this.nombre = nombre;
        this.salario = salario;
    }



    public void tiempoextra(int horas) {
        //codigo de que puesto tendria el empleado
    };
    public void salario(double salario) {
        //codigo de salario para empleado
    }
    public void fondosdereserva(double tiempotrabajando) {
        //codigo

    }
```

```java
package Problema;
import java.util.ArrayList;
public class ISPProblema {
    public static void main(String[] args) {
        List<Empleados> empl = new ArrayList<>();
        empl.add(new Empleados("Katherine",450));


        for (Empleados empleado : empl ) {
            empleado.tiempoextra();
            empleado.salario();
            empleado.fondosdereserva();

        }
    }
}
```

Solución

```java
package Solucion;

public interface EmpleadoEstudiante extends TiempoExtra {
     // Métodos especificos para empleadores que trabajan medio ti
    public void horasdetrabajo();
    public void tiempoextra(int tiempoenlaempresa);

}
```

```java
package Solucion;

public interface FondReserva {
    public void fondosdereserva(double tiempoenlaempresa);
}
```

```java
package Solucion;

public interface Salario {
    public void pagsalario(double monto);
}
```

```java
package Solucion;

public interface TiempoExtra {
    public void tiempoextra(int tiempoenlaempresa);
}
```

```java
package Solucion;

import java.util.ArrayList;


public class ISPSolucion {
    public static void main(String[] args) {
        List<EmpEst> empleados = new ArrayList<>();
        empleados.add(new EmpEst());


        for (EmpEst empleado : empleados) {
            empleado.horasdetrabajo();
            empleado.tiempoextra(3);

        }
    }
}
```

```java
package Solucion;

public class EmpEst implements EmpleadoEstudiante{
    //metodos de la clase EmpEst

    public void horasdetrabajo() {

        //
    }

    @Override
    public void tiempoextra(int tiempoenlaempresa) {
        // TODO Auto-generated method stub

    }
}
```

Dependency Inversion Principle (DIP)

Link ejemplo: https://javatechonline.com/solid-principles-the-dependency-inversion-principle/

Adaptación

Problema

```java
package Problema;

public class MaquinaExpendedora {

    //nuestra clase depende de una sola clase
    //sin embargo esta podria ser de distinto tipo no solo moneda sino billete
    //o incluso tarjeta
    Moneda dinero;

    public void ingresaDinero(Moneda dinero) {

    }

    public void botarProducto() {

    }

    public void devuelveCambio() {

    }

}
```

```java
package Problema;

public class Moneda {

    public Moneda() {

    }

}
```

Solución

```java
package Solucion;

import Problema.Moneda;

public class MaquinaExpendedora {

    //nuestra clase ahora depende de una interfaz que simplemente
    //se podria ingresar cualquier implementacion que haya hecho
    //y es mas facil.
    Dinero dinero;

    public void ingresaDinero(Dinero dinero) {

    }

    public void botarProducto() {

    }

    public void devuelveCambio(Dinero dinero) {

    }
}
```

```java
package Solucion;

public interface Dinero {

    public Dinero entregarDinero();

}
```

```java
package Solucion;

public class Moneda implements Dinero {

    @Override
    public Dinero entregarDinero() {
        // TODO Auto-generated method stub
        return null;
    }

}
```

```java
package Solucion;

public class Billete implements Dinero{

    @Override
    public Dinero entregarDinero() {
        // TODO Auto-generated method stub
        return null;
    }


}
```

Sección B

Clase Notificación

- **Principio SOLID violado:** Open-Close Principle (OCP)
- **Explicación SOLID violado:** El método notificar no permite extender por ende no existe polimorfismo
- **Solución aplicando principios SOLID**

```java
package Taller;

public interface TipoNotificacion {

    public void notificar();



}
```

```java
package Taller;

public class SMS implements TipoNotificacion{
    String destinatario;
    String emisor;
    String motivo;

    public void notificar() {
        System.out.print("Enviado por: " + destinatario + " para" + emisor + motivo);
    }
}
```

Clase Pago

- **Principio SOLID violado:** Single-Responsability Principle(SRP)
- **Explicación SOLID violado:** Esta clase incumple SRP ya que tiene 3 razones para ir cambiando lo cual impide que dicha clase se mantenga.
- **Solución aplicando principios SOLID**

```java
package Taller;

public interface Cobrable {
    public void realizarCobro();

}
```

```java
package Taller;

public interface Impuestos {
    public void calcularImpuestosFactura();
}
```

```java
package Taller;

public interface Factura {
    public void generarFactura();
}

```

Clase PagoPaypal

- **Principio SOLID violado:** Liskov Substitution Principle (LSP)
- **Explicación SOLID violado:** Al extender de Pago y que solo sea ese método de pago es muy limitado ya que si el pago no es mediante Paypal no se puede realizar el pago.
- **Solución aplicando principios SOLID**

```java
package Taller;

public interface Pago {
    //ahora puede ser implementado por cualquier metodo de pago
    public void pagar();
}
```

```java
package Taller;

public class Paypal implements Pago{

    @Override
    public void pagar() {
        // TODO Auto-generated method stub

    }

}
```

Clase Compra

- **Principio SOLID violado:** Dependency Inversion Principle (DIP)
- **Explicación SOLID violado:** El parámetro de esta quemado. Pago debe ser una interfaz para que pueda existir medios de pagos por ende debe implementar varios métodos de pago.
- **Solución aplicando principios SOLID**

```java
package Taller;

public interface Pago {

    public void pagar();
}
```

```java
package Taller;

import java.util.List;

public class Compra {
    //Cada atributo va a ser de tipo abstracto
    //asi va a poder recibir de cualquier que la clase/interfaz abstracta implemente

    private Pago pago;
    private List elementos;


}
```

Link del repositorio:

https://github.com/carana08/Taller03.git