PRINCIPIOS SOLID

Conjunto de reglas y buenas prácticas que se deben seguir para diseñar un software.

Single Responsibility Principle
Open-Closed Principle
Liskov Substitution Principle
Interface Segregation Principle
Dependency Inversion Principle

Principio de responsabilidad única: Una clase o módulo solo debería hacer una sola cosa, una clase solo debe tener una razón para cambiar.

```
public class ManupularTexto {
   private String texto;

   public ManupularTexto() {
   }

   public ManupularTexto(String texto) {
      this.texto = texto;
   }

   public String getTexto() {
      return texto;
   }

   public void concatenarTexto(String nuevoTexto) {
      texto.concat(nuevoTexto);
   }

   public String reemplazarPalabra(String palabra, String nuevaPalabra) {
      if(texto.contains(palabra)) {
```

```
            texto = texto.replace(palabra, nuevaPalabra);
            return texto;
        } else {
            return "el texto no contiene la palabra " + palabra;
        }
    }
}
```

Principio abierto-cerrado: Una clase debe estar abierta para su extensión y cerrada a ser modificada.

```
public interface Calculadora {
    void realizarOperacion(int num1, int num2);
}

public class Suma implements Calculadora{
    @Override
    public void realizarOperacion(int num1, int num2) {
        System.out.println(num1 + num2);
    }
}

public class Resta implements Calculadora{
    @Override
    public void realizarOperacion(int num1, int num2) {
        System.out.println(num1 - num2);
    }
}
```

Principio de sustitución de Liskov: Subclases deberían ser sustituibles por sus clases padre.

```
public class Cuenta {
    private int saldo;

    public void depositar(int saldoDepositado) {
        saldo = saldo + saldoDepositado;
        System.out.println(saldo);
    }
```

```
    public void retirar(int saldoRetirado) {
        saldo = saldo - saldoRetirado;
        System.out.println(saldo);
    }

}

public class Ahorros extends Cuenta{
}

public class Corriente extends Cuenta{
}

public class Main {
    public static void main(String[] args) {
        Cuenta ahorros = new Ahorros();
        Cuenta corriente = new Corriente();
        ahorros.depositar(2000);
        ahorros.retirar(500);
        corriente.depositar(3000);
        corriente.retirar(200);
    }
}
```

Principio de segregación de interfaces: Es mejor tener varias interfaces específicas a una general ya que una clase no debe ser forzada a implementar un método que no necesita.

```
public interface Nadador {
    void nadar();
}

public interface Corredor {
    void correr();
}

public interface Volador {
    void volar();
}

public class Avestruz implements Corredor{
    @Override
```

```
    public void correr() {
        System.out.println("Avestruz corriendo.");
    }
}

public class Pato implements Nadador, Corredor, Volador{
    @Override
    public void correr() {
        System.out.println("Pato corriendo");
    }

    @Override
    public void nadar() {
        System.out.println("Pato nadando");
    }

    @Override
    public void volar() {
        System.out.println("Pato volando");
    }
}
```

Principio de inversión de dependencias: Una clase debería depender de su interfaz y no de su implementación.

```
public class Empleado {
    int id;
    String nombre;

    public Empleado(int id, String nombre) {
        this.id = id;
        this.nombre = nombre;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

```
    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}

public interface IEmpleado {
    Empleado buscarEmpleado(int id);
    void guardarEmpleado(Empleado empleado);
}

public class EmpleadoImpl implements IEmpleado{
    @Override
    public Empleado buscarEmpleado(int id) {
        return new Empleado(id, "Alex");
    }

    @Override
    public void guardarEmpleado(Empleado empleado) {
        System.out.println("Nombre empleado: " + empleado.getNombre());
    }
}

public class Operaciones {

    private final IEmpleado empleado;

    public Operaciones(IEmpleado empleado) {
        this.empleado = empleado;
    }

    Empleado buscarEmpleado(int id) {
        return empleado.buscarEmpleado(id);
    }

    void guardarEmpleado(Empleado empl){
        empleado.guardarEmpleado(empl);
    }
```

```
}
```