**Universidad de las Fuerzas Armadas ESPE**

**Departamento:** Ciencias de la Computación

**Carrera:** Ingeniería de Software

**Taller académico Nª:** 3

**1. Información General**

* **Asignatura:** Análisis y Diseño de Software
* **Apellidos y nombres de los estudiantes:** Pamela Chipe, Carlos Jaya, Elkin Pabón
* **NRC:** 23305
* **Fecha de realización:** 07/07/2025

**2. Objetivo del Taller y Desarrollo**

**Objetivo del Taller:**

Comprender el Principio de Responsabilidad Única (SRP) del conjunto SOLID mediante el análisis comparativo de ejemplos de clases en un ejercicio CRUD en Java, identificando responsabilidades mezcladas y reconociendo cómo una adecuada separación de responsabilidades mejora la claridad, mantenibilidad y diseño del software.

**Desarrollo:**

1. **¿Qué es SRP?**

SRP significa Single Responsibility Principle (Principio de Responsabilidad Única), que establece que una clase debe tener una única razón para cambiar, es decir, debe estar encargada de una sola funcionalidad o responsabilidad dentro del sistema. Este principio permite que el código sea más comprensible, fácil de mantener y menos propenso a errores.

**Análisis de las clases del ejercicio CRUD del estudiante:**

**Clase Modelo (Estudiante)**

* Se encarga exclusivamente de representar los datos del estudiante (como nombre e identificación).
* No contiene lógica adicional como impresión o almacenamiento.
* Cumple SRP porque su única razón para cambiar es una modificación en la estructura de los datos del estudiante.

**Clase DAO (EstudianteDAO)**

* Su única función es manejar la persistencia de los datos, es decir, guardar al estudiante en la base de datos.
* Está completamente separada del modelo de datos y de la presentación.
* Cumple SRP porque solo cambiaría si cambia la forma de conectarse o guardar en la base de datos.

**Clase Vista (EstudiantePrinter)**

* Se encarga únicamente de mostrar la información del estudiante en pantalla (presentación).
* No tiene acceso a la lógica de negocio ni a la persistencia.
* Cumple SRP porque solo cambiaría si se desea modificar el formato o estilo de impresión.

1. **Ejemplo que NO cumple SRP**

La siguiente clase mezcla varias responsabilidades

public class Estudiante {  
 private int id;  
 private String nombre;  
  
 public Estudiante(int id, String nombre) {  
 this.id = id;  
 this.nombre = nombre;  
 }  
  
 public void guardarEnBD() {  
 // Código que guarda en la base de datos  
 System.out.println("Guardando en BD...");  
 }  
  
 public void imprimir() {  
 // Código que imprime en consola  
 System.out.println("Estudiante: " + nombre);  
 }  
 }

La clase Estudiante presentada en este ejemplo viola el principio SRP porque concentra múltiples responsabilidades en una sola estructura, lo que la hace difícil de mantener y escalar.

1. **Modelar los datos del estudiante**

La clase almacena los atributos id y nombre, y por lo tanto representa la estructura de un estudiante.

1. **Guardar el estudiante en la base de datos**

El método guardarEnBD() implementa la lógica para almacenar los datos del estudiante en una base de datos. Esta operación debería estar delegada a una clase especializada en acceso a datos.

1. **Imprimir la información del estudiante**

A través del método imprimir(), la clase realiza tareas de presentación o salida de datos, función que también debería estar separada.

Esta clase tiene más de una razón para cambiar:

* Si cambia la forma de representar al estudiante (por ejemplo, agregar nuevos campos), se modifica el modelo.
* Si cambia la tecnología de almacenamiento (por ejemplo, de archivos a base de datos), se modifica la lógica de persistencia.
* Si cambia el formato de presentación (por ejemplo, de texto plano a HTML), se modifica la salida.

1. **Anàlisis del Problema**

Esta clase tiene 3 responsabilidades distintas

1. Modelar los datos del estudiante
2. Guardar el estudiante en la base de datos (persistencia)
3. Imprimir la información del estudiante (presentación)
4. **Ejemplo que SÍ cumple SRP**

Ahora se separan las responsabilidades en clases distintas:

1. Clase Modelo (solo datos):

public class Estudiante {  
 private int id;  
 private String nombre;  
  
 public Estudiante(int id, String nombre) {  
 this.id = id;  
 this.nombre = nombre;  
 }  
  
 // Getters y Setters  
 public int getId() { return id; }  
 public String getNombre() { return nombre; }  
 }

2. Clase DAO (persistencia):

public class EstudianteDAO {  
 public void guardar(Estudiante e) {  
 // Código para guardar en BD  
 System.out.println("Guardando estudiante en la BD...");  
 }  
 }

3. Clase Vista (presentación):

public class EstudiantePrinter {  
 public void imprimir(Estudiante e) {  
 System.out.println("Estudiante: " + e.getNombre());  
 }  
 }

1. **Ventajas de aplicar SRP**

(analiza para responder las siguientes preguntas).

a. Si cambias cómo se imprimen los datos, no tocas la clase **Estudiantes**

b. Si cambias cómo se guardan, tampoco tocas el **modelo ni la vista**

Cada clase tiene UNA SOLA razón para cambiar

Rúbrica de Evaluación

| Criterio | Puntaje Máximo |
| --- | --- |
| Explica con claridad qué es SRP | 4 puntos |
| Identifica correctamente las responsabilidades mezcladas | 4 puntos |
| Muestra correctamente el ejemplo que sí cumple SRP | 4 puntos |
| Analiza ventajas de aplicar SRP | 4 puntos |
| Presenta el contenido de forma clara y ordenada | 4 puntos |
| TOTAL SOBER 20 PTOS |  |

**3. Referencias (Norma APA 7.0)**

* Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson Education.
* Pressman, R. S., & Maxim, B. R. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill Education.
* Beck, K., & Andres, C. (2004). *Extreme Programming Explained: Embrace Change* (2nd ed.). Addison-Wesley.