PATRONES DE DISEÑO

Mateo LLumigusin, David Asmal, Richard Gualotuña

Departamento de Ciencias de la Computación, Universidad de las Fuerzas Armadas ESPE

NRC: 23305

TUTORA Ing. Jenny Alexandra Ruíz Robalino

Fecha: 11/06/2025

Introducción

En esta práctica se analiza la aplicación del Principio de Responsabilidad Única (SRP), el primero del conjunto de principios SOLID, fundamentales en la programación orientada a objetos. SRP establece que una clase debe tener una única razón para cambiar, lo que se traduce en que debe tener una única responsabilidad dentro del sistema.

El objetivo de este informe es evaluar cómo se cumple (o no) este principio en un ejemplo concreto: un sistema CRUD (Crear, Leer, Actualizar y Eliminar) de la entidad Estudiante desarrollado en Java. Este análisis no solo refuerza los conocimientos de diseño orientado a objetos, sino que también permite desarrollar criterios sólidos para mejorar la calidad y la escalabilidad de nuestras aplicaciones.

1. ¿Qué es el SRP?

El Single Responsibility Principle (SRP) define que una clase debe tener una única responsabilidad, es decir, solo una razón para cambiar. Cuando una clase asume múltiples funciones (por ejemplo, representar datos, manejar persistencia y controlar la presentación), se vuelve más difícil de mantener, probar y escalar.

Ventajas de SRP:

- Clases pequeñas y enfocadas.
- Menor acoplamiento.
- Mayor facilidad para realizar cambios sin afectar otras partes del sistema.
- Mejor organización y reutilización de código.

2. Ejemplo que NO cumple con SRP

java

```
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);
    }
}
```

Análisis

Esta clase está violando SRP porque tiene tres responsabilidades diferentes:

- Modelo de datos: contiene atributos (id, nombre) que representan al estudiante.
- Persistencia: el método guardarEnBD() intenta realizar una operación de acceso a datos.
- Presentación: el método imprimir() tiene lógica para mostrar la información en consola.

Al mezclar estas responsabilidades en una sola clase, cualquier cambio en la forma en que se imprime, o cómo se guarda, obliga a modificar esta clase, lo que genera acoplamiento innecesario y baja cohesión.

3. **Ejemplo que SÍ cumple con SRP**

Para cumplir con SRP, se han separado las responsabilidades en tres clases distintas:

Modelo de Datos

java

```
public class Estudiante {
    private int id;
    private String nombre;

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

    public int getId() { return id; }
    public String getNombre() { return nombre; }
}
```

Clase de Persistencia (DAO)

java

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

}

Clase de Presentación

java

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

Resultado

Ahora cada clase tiene una sola razón para cambiar:

- El modelo Estudiante cambia solo si cambian los datos.
- EstudianteDAO cambia si se modifica la lógica de persistencia.
- EstudiantePrinter cambia si se modifica la lógica de presentación.

Ventajas de aplicar SRP

Situación de cambio	**Afecta solo a…**	**No se modifica…**
Cambio en impresión	EstudiantePrinter	Estudiante, EstudianteDAO
Cambio en guardado	EstudianteDAO	Estudiante, EstudiantePrinter
Cambio en datos	Estudiante	DAO, Vista

Conclusiones

- El SRP es fundamental para diseñar sistemas robustos. Permite que cada clase cumpla un único propósito, lo cual simplifica el desarrollo, mantenimiento y prueba de software.

- En el ejemplo del CRUD de Estudiante, se evidencia que una clase mal diseñada (con múltiples responsabilidades) puede traer consecuencias negativas en el crecimiento del sistema.
- La versión que cumple SRP demuestra una estructura clara y desacoplada, donde cada componente tiene un rol específico y definido.

Recomendaciones

- Evitar mezclar responsabilidades en una sola clase. Separar siempre lógica de negocio, datos y presentación.
- Utilizar SRP desde el inicio del desarrollo de un sistema. Refactorizar más adelante es más costoso.
- Aplicar SRP junto con otros principios SOLID, como OCP (Open/Closed) y DIP (Dependency Inversion), para lograr una arquitectura profesional.