Universidad de las Fuerzas Armadas - ESPE

Departamento de Ciencias de la Computación

Carrera de Ingeniería de Software

Análisis y Diseño de Software - NRC:22426

Trabajo:

U2T5 - Principio del SOLID

Grupo: 4

Integrantes:

Diego Casignia

Anthony Villarreal

Javier Ramos

Profesora: Ing. Jenny Ruiz

TRABAJO EN GRUPO (5ptos)

Ejemplo de SRP (Single Responsibility Principle) en Java PATRÓN SOLID

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 enfocada en una sola responsabilidad dentro del sistema. Esto implica que cada clase, módulo o función debe encargarse exclusivamente de una parte específica del comportamiento del programa, lo que facilita su mantenimiento, comprensión, prueba y reutilización.

2. Ejemplo que NO cumple SRP

La siguiente clase mezcla varias responsabilidades

Esta clase viola el Principio de Responsabilidad Única (SRP) porque mezcla tres responsabilidades distintas en una sola clase. Primero, actúa como modelo de datos, ya que representa al estudiante mediante los atributos id y nombre. Sin embargo, también incluye la responsabilidad de persistencia, al implementar el método guardarEnBD(), lo cual debería ser manejado por una clase DAO especializada en el acceso a datos. Además, incorpora la presentación de la información con el método imprimir(), que debería estar en una clase separada dedicada a mostrar datos, como una vista o presentador. Esta combinación de roles en una sola clase contradice el SRP, ya que hay más de una razón para que esta clase cambie en el futuro.

(Identifica cuales son las responsibilities ej presenter datos,)

```
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);
        }
}
```

3. Anàlisis del Problema

Esta clase Main tiene 3 responsabilidades distintas
1. Gestión de la interfaz de usuario
2. Control del flujo de la aplicación
3. Invocar la lógica de negocio

```
package ui;

import controller.EstudianteController;

import model.Estudiante;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        EstudianteController controller = new EstudianteController();

        Scanner sc = new Scanner(System.in);

        int opcion;

        do {

            // Mostrar el menú por consola

            System.out.println("\n--- MENU CRUD ESTUDIANTES ---");

            System.out.println("1. Agregar estudiante");

            System.out.println("2. Mostrar todos los estudiantes");

            System.out.println("3. Buscar estudiante por ID");

            System.out.println("4. Actualizar estudiante");

            System.out.println("5. Eliminar estudiante");

            System.out.println("0. Salir");

            System.out.print("Seleccione una opción: ");

            opcion = sc.nextInt();

            sc.nextLine();

            switch (opcion) {

                case 1:

                    System.out.print("ID: ");
```

```
                int id = sc.nextInt();
                sc.nextLine();
                System.out.print("Apellidos: ");
                String apellidos = sc.nextLine();
                System.out.print("Nombres: ");
                String nombres = sc.nextLine();
                System.out.print("Edad: ");
                int edad = sc.nextInt();
                controller.crearEstudiante(id, apellidos, nombres, edad);
                System.out.println("Estudiante agregado correctamente.");
                break;
            case 2:
                System.out.println("\nLista de estudiantes:");
                for (Estudiante e : controller.obtenerTodos()) {
                    System.out.println(e.getId() + " - " + e.getApellidos() + " " + e.getNombres() + " - Edad: " +
e.getEdad());
                }
                break;
            case 3:
                System.out.print("Ingrese ID del estudiante: ");
                int idBuscar = sc.nextInt();
                Estudiante e = controller.buscarEstudiante(idBuscar);
                if (e != null) {
            case 4:
                System.out.print("ID del estudiante a actualizar: ");
                int idActualizar = sc.nextInt();
                sc.nextLine();
```

```
                System.out.print("Nuevos apellidos: ");
                String nuevosApellidos = sc.nextLine();
                System.out.print("Nuevos nombres: ");
                String nuevosNombres = sc.nextLine();
                System.out.print("Nueva edad: ");
                int nuevaEdad = sc.nextInt();
                boolean actualizado = controller.actualizarEstudiante(idActualizar, nuevosApellidos,
nuevosNombres, nuevaEdad);
                System.out.println(actualizado ? "Estudiante actualizado." : "No se encontró el estudiante.");
                break;
            case 5:
                System.out.print("ID del estudiante a eliminar: ");
                int idEliminar = sc.nextInt();
                boolean eliminado = controller.eliminarEstudiante(idEliminar);
                System.out.println(eliminado ? "Estudiante eliminado." : "No se encontró el estudiante.");
                break;
            case 0:
                System.out.println("Saliendo del sistema...");
                break;
            default:
                System.out.println("Opción inválida.");
        }
    } while (opcion != 0);
    sc.close();
  }
}
```

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());
        }
}
```

En esta solución, cada clase tiene una única responsabilidad:

- La clase Estudiante solo maneja los datos del estudiante (atributos y getters).
- La clase EstudianteDAO se encarga exclusivamente de la persistencia (guardar en base de datos).
- La clase EstudiantePrinter solo maneja la presentación de los datos (imprimir en consola).

Esta separación cumple perfectamente con el SRP, ya que cada clase tiene una única razón para cambiar y una única responsabilidad bien definida.

5. Ventajas de aplicar SRP

a. Si cambias cómo se imprimen los datos, no tocas la clase que gestiona la lógica de negocio, solo modificas la clase encargada de la presentación.

b. Si cambias cómo se guardan, tampoco tocas el controlador ni la interfaz de usuario, solo modificas la clase encargada de la persistencia (por ejemplo, el EstudianteDAO).

c. Cada clase tiene UNA SOLA razón para cambiar, lo que mejora la claridad y el mantenimiento del código.

Rúbrica de Evaluation

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	