Programación orientada a objetos Individual - POO - Unidad 3
Actividad: Taller sobre Principios SOLID en Java - Responsabilidad Única
Tutor: JOHN CARLOS ARRIETA ARRIETA DOCENTE

Taller: Principio de Responsabilidad Única (SRP - Single Responsibility Principle)

Objetivo del Taller:

En este taller, los estudiantes aprenderán el **Principio de Responsabilidad Única (SRP)** en el diseño orientado a objetos. El SRP establece que cada clase debe tener una sola responsabilidad o razón para cambiar. Este taller proporciona una definición formal y una explicación en lenguaje cotidiano, así como sus beneficios, desventajas, conceptos relacionados, ejemplos de aplicación y violaciones, y ejercicios propuestos para afianzar el conocimiento.

---

1. Definición Formal del SRP:

El **Principio de Responsabilidad Única (Single Responsibility Principle)** indica que **una clase debe tener una sola responsabilidad o razón para cambiar**. En otras palabras, cada clase debe encargarse de una única tarea o propósito.

- **Definición Técnica:** "Cada clase debe tener una sola razón para cambiar."

Imagina que tienes un equipo (aplicación orientada a objetos) de trabajo donde cada persona (clase o interfaz) tiene una tarea (método) específica. Si cada miembro del equipo se especializa en una única tarea, será más fácil gestionar los cambios y mejorar el rendimiento del equipo (cohesión). De igual forma, una clase en programación debe encargarse de una sola tarea para ser fácil de mantener y entender.

---

2. Beneficios del SRP:

- **Mantenimiento Simplificado:** Si cada clase tiene una única responsabilidad, es más fácil realizar cambios sin afectar otras funcionalidades.
- **Pruebas Unitarias Más Eficientes:** Al tener una sola responsabilidad, las clases se pueden probar de manera aislada.
- **Mayor Cohesión:** Cada clase es responsable de una única tarea, lo que mejora la claridad del código y la cohesión interna.
- **Facilita la Reutilización:** Las clases especializadas se pueden reutilizar en diferentes contextos sin modificaciones.

---

3. Desventajas del SRP:

- **Incremento de la Cantidad de Clases:** Aplicar el SRP puede llevar a tener muchas clases pequeñas, lo que puede resultar abrumador si no se organiza adecuadamente.
- **Posible Complejidad en el Diseño:** La separación de responsabilidades puede aumentar la complejidad inicial del diseño, especialmente si no se tiene experiencia en el principio.

---

4. Conceptos Relacionados que se Deben Dominar:

- **Cohesión:** Medida de cuán relacionadas están las responsabilidades dentro de una clase.
- **Acoplamiento:** Grado de dependencia entre clases.
- **Encapsulamiento:** Ocultación de los detalles internos de una clase.
- **Delegación:** Enviar una responsabilidad a otra clase más adecuada para manejarla.

---

5. Ejemplos de Aplicación del SRP:

Ejemplo 1: Gestión de Usuarios en una Aplicación

- **Situación Inicial:** Una clase `Usuario` maneja autenticación, validación y generación de informes.
- **Aplicación del SRP:** Separar las responsabilidades en clases individuales.

```
// Clase Usuario (solo datos del usuario)
public class Usuario {
    private String nombre;
    private String contrasena;

    public Usuario(String nombre, String contrasena) {
        this.nombre = nombre;
        this.contrasena = contrasena;
    }
```

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

```
// Clase AutenticacionService (responsable de la autenticación)
public class AutenticacionService {
    public boolean autenticar(Usuario usuario, String contrasena) {
        return usuario.getNombre().equals("admin") && contrasena.equals("1234");
    }
}
```

```
// Clase InformeService (responsable de la generación de informes)
public class InformeService {
    public void generarInformeUsuario(Usuario usuario) {
        System.out.println("Generando informe para " + usuario.getNombre());
    }
}
```

Ejemplo 2: Procesamiento de Pedidos

- **Situación Inicial:** Una clase `Pedido` maneja la lógica de cálculo del total, impresión del recibo y envío del pedido.
- **Aplicación del SRP:** Separar las responsabilidades en clases específicas.

```
// Clase Pedido (solo datos del pedido)
public class Pedido {
    private double total;

    public Pedido(double total) {
        this.total = total;
    }

    public double getTotal() {
        return total;
    }
}
```

```
// Clase CalculoTotalService (responsable del cálculo del total)
public class CalculoTotalService {
    public double calcularImpuestos(Pedido pedido) {
        return pedido.getTotal() * 0.15;
    }
}
```

```
// Clase EnvioService (responsable del envío de pedidos)
public class EnvioService {
    public void enviarPedido(Pedido pedido) {
        System.out.println("Enviando pedido con total: " + pedido.getTotal());
    }
}
```

Ejemplo 3: Sistema de Notificaciones

Programación orientada a objetos Individual - POO - Unidad 3
Actividad: Taller sobre Principios SOLID en Java - Responsabilidad Única
Tutor: JOHN CARLOS ARRIETA ARRIETA DOCENTE

- **Situación Inicial:** Una clase `Notificacion` maneja la lógica para enviar notificaciones por correo y SMS.
- **Aplicación del SRP:** Dividir la clase en dos clases separadas para correo y SMS.

java

Copiar código

```java
// Clase EmailService (responsable de las notificaciones por correo)
public class EmailService {
    public void enviarEmail(String destinatario, String mensaje) {
        System.out.println("Enviando email a: " + destinatario);
        System.out.println("Mensaje: " + mensaje);
    }
}
```

```java
// Clase SMSService (responsable de las notificaciones por SMS)
public class SMSService {
    public void enviarSMS(String numero, String mensaje) {
        System.out.println("Enviando SMS a: " + numero);
        System.out.println("Mensaje: " + mensaje);
    }
}
```

---

6. Ejemplos de Violación del SRP:

Ejemplo 1: Clase `Empleado` con Responsabilidades Múltiples

- **Violación:** La clase `Empleado` tiene métodos para la gestión de su información personal, cálculo de salario y generación de informes.

```java
public class Empleado {
    private String nombre;
    private double salario;

    public double calcularSalarioMensual() {
        return salario * 12;
    }

    public void generarInforme() {
        System.out.println("Generando informe para: " + nombre);
    }

    public void guardarEnBaseDeDatos() {
        System.out.println("Guardando empleado en la base de datos...");
    }
}
```

Problema: La clase `Empleado` tiene tres responsabilidades diferentes: cálculos, generación de informes y persistencia.

Ejemplo 2: Clase `Factura` con Responsabilidades Múltiples

- **Violación:** La clase `Factura` maneja la lógica de impresión, cálculo e interacción con la base de datos.

```java
public class Factura {
    private double monto;

    public double calcularImpuestos() {
```

```
        return monto * 0.12;
    }

    public void imprimirFactura() {
        System.out.println("Imprimiendo factura...");
    }

    public void guardarFactura() {
        System.out.println("Guardando factura en la base de datos...");
    }
}
```

Problema: La clase `Factura` tiene responsabilidades relacionadas con el cálculo, la impresión y la persistencia.

---

7. Ejercicios Propuestos

Ejercicio 1: Refactorización de una Clase `Libro`

1. Define una clase `Libro` que contenga métodos relacionados con el manejo de su información, generación de reportes y persistencia.
2. Identifica y separa las responsabilidades en clases específicas para aplicar el SRP.

Ejercicio 2: Separación de Responsabilidades en una Clase `Producto`

1. Define una clase `Producto` que maneje datos del producto, generación de etiquetas y cálculo de precios.
2. Refactoriza la clase para dividir las responsabilidades utilizando el SRP.

Ejercicio 3: Separar la Lógica de Autenticación en un Sistema

1. Define una clase `Usuario` con datos de usuario, autenticación y validación.
2. Refactoriza la clase utilizando el SRP para crear servicios especializados para autenticación y validación.

Conclusión del Taller:

Este taller ha proporcionado una comprensión profunda del **Principio de Responsabilidad Única (SRP)** y cómo aplicarlo para crear clases más cohesivas y de fácil mantenimiento. Al aplicar el SRP, los estudiantes aprenderán a estructurar sus clases de una manera que facilite la comprensión, las pruebas y la evolución del código.