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

Taller: Principio de Segregación de Interfaces (ISP - Interface Segregation Principle)

Objetivo del Taller:

En este taller, los estudiantes aprenderán el **Principio de Segregación de Interfaces (ISP)**, uno de los principios fundamentales del diseño orientado a objetos en SOLID. Este principio establece que **los clientes no deben depender de interfaces que no utilizan**. El taller aborda definiciones formales y cotidianas, sus beneficios, desventajas, conceptos previos, ejemplos correctos e incorrectos, y ejercicios propuestos para afianzar el conocimiento.

---

1. Definición Formal del ISP:

El **Principio de Segregación de Interfaces (Interface Segregation Principle)** establece que **una clase no debe ser forzada a depender de métodos que no utiliza**. En otras palabras, es mejor tener varias interfaces pequeñas y específicas que una sola interfaz grande y general.

- **Definición Técnica:** "Los clientes no deben ser forzados a depender de interfaces que no utilizan."

Imagina que tienes un restaurante (aplicación OO) y a cada empleado (clase) se le asignan tareas específicas basadas en su rol. No tendría sentido exigirle al cocinero (clase `Cocinero`) que limpie las mesas (interfaz `Limpieza`) o pedirle al camarero (clase `Camarero`) que cocine los platos (interfaz `Cocina`). Cada empleado debe cumplir con las tareas que corresponden a su rol (interfaces específicas), y no forzarlos a asumir responsabilidades que no les corresponden.

---

2. Beneficios del ISP:

- **Reduce el Acoplamiento:** Al tener interfaces más específicas, las clases dependen solo de los métodos que realmente utilizan, lo que disminuye el acoplamiento.
- **Facilita el Mantenimiento:** Si se realizan cambios en una interfaz, estos afectarán a menos clases, facilitando el mantenimiento del sistema.
- **Permite la Reutilización de Código:** Las interfaces específicas se pueden reutilizar en diferentes contextos sin forzar la implementación de métodos no necesarios.

---

3. Desventajas del ISP:

- **Incremento de Interfaces:** El ISP puede llevar a la creación de múltiples interfaces pequeñas, lo que puede hacer el diseño más complejo si no se gestiona correctamente.
- **Mayor Complejidad Inicial:** Aplicar el ISP correctamente puede requerir un análisis y diseño detallado para identificar y separar las responsabilidades.

---

4. Conceptos Relacionados que se Deben Dominar:

- **Interfaz:** Definición de un contrato que las clases implementan.
- **Acoplamiento:** Grado de dependencia entre clases o módulos.
- **Cohesión:** Relación entre las responsabilidades dentro de una interfaz o clase.
- **Polimorfismo:** Capacidad de una clase para implementar diferentes versiones de un método definido en una interfaz.

---

5. Ejemplos de Aplicación del ISP:

Ejemplo 1: Sistema de Empleados en un Restaurante

- **Situación Inicial:** Una interfaz `Empleado` define métodos para cocinar, limpiar, y atender mesas, pero algunos empleados solo necesitan implementar uno o dos de estos métodos.
- **Aplicación del ISP:** Dividir la interfaz `Empleado` en interfaces más específicas.

```
// Interfaz Cocinero
public interface Cocinero {
    void cocinar();
}
```

```
// Interfaz Camarero
```

```java
public interface Camarero {
    void atenderMesas();
}
```

```java
// Interfaz PersonalLimpieza
public interface PersonalLimpieza {
    void limpiar();
}
```

```java
// Clase CocineroEmpleado
public class CocineroEmpleado implements Cocinero {
    @Override
    public void cocinar() {
        System.out.println("El cocinero está preparando la comida.");
    }
}
```

```java
// Clase CamareroEmpleado
public class CamareroEmpleado implements Camarero {
    @Override
    public void atenderMesas() {
        System.out.println("El camarero está atendiendo las mesas.");
    }
}
```

Explicación: Cada clase implementa únicamente los métodos de la interfaz que necesita, siguiendo el ISP.

Ejemplo 2: Sistema de Notificaciones

- **Situación Inicial:** Una interfaz `Notificacion` define métodos para enviar correos y mensajes de texto, pero no todos los servicios de notificación necesitan ambos métodos.
- **Aplicación del ISP:** Dividir la interfaz `Notificacion` en interfaces más específicas.

```java
// Interfaz EnvioEmail
public interface EnvioEmail {
    void enviarEmail(String destinatario, String mensaje);
}
```

```java
// Interfaz EnvioSMS
public interface EnvioSMS {
    void enviarSMS(String numero, String mensaje);
}
```

```java
// Clase ServicioNotificacionEmail
public class ServicioNotificacionEmail implements EnvioEmail {
    @Override
    public void enviarEmail(String destinatario, String mensaje) {
        System.out.println("Enviando email a: " + destinatario);
        System.out.println("Mensaje: " + mensaje);
    }
}
```

```java
// Clase ServicioNotificacionSMS
public class ServicioNotificacionSMS implements EnvioSMS {
    @Override
    public void enviarSMS(String numero, String mensaje) {
        System.out.println("Enviando SMS a: " + numero);
        System.out.println("Mensaje: " + mensaje);
    }
}
```

Explicación: Cada clase implementa únicamente las interfaces que necesita, lo que evita la implementación de métodos innecesarios.

---

6. Ejemplos de Violación del ISP:

Ejemplo 1: Interfaz `Empleado` con Métodos Innecesarios

- **Violación:** La interfaz `Empleado` tiene métodos que no son relevantes para todos los empleados.

```
// Interfaz Empleado
public interface Empleado {
    void cocinar();
    void limpiar();
    void atenderMesas();
}
```

```
// Clase CocineroEmpleado
public class CocineroEmpleado implements Empleado {
    @Override
    public void cocinar() {
        System.out.println("El cocinero está cocinando.");
    }

    @Override
    public void limpiar() {
        // Implementación vacía, ya que el cocinero no debería limpiar.
    }

    @Override
    public void atenderMesas() {
        // Implementación vacía, ya que el cocinero no debería atender mesas.
    }
}
```

Problema: La clase `CocineroEmpleado` está obligada a implementar métodos que no le corresponden, lo que viola el ISP.

Ejemplo 2: Interfaz `Notificacion` con Métodos Irrelevantes

- **Violación:** La interfaz `Notificacion` tiene métodos que no son relevantes para todos los servicios de notificación.

```
// Interfaz Notificacion
public interface Notificacion {
    void enviarEmail(String destinatario, String mensaje);
    void enviarSMS(String numero, String mensaje);
}
```

```
// Clase ServicioEmail
public class ServicioEmail implements Notificacion {
    @Override
    public void enviarEmail(String destinatario, String mensaje) {
        System.out.println("Enviando email a: " + destinatario);
        System.out.println("Mensaje: " + mensaje);
    }

    @Override
    public void enviarSMS(String numero, String mensaje) {
        // Implementación vacía, ya que el servicio de email no envía SMS.
    }
}
```

Problema: La clase `ServicioEmail` está obligada a implementar un método de SMS que no necesita, lo que viola el ISP.

---

7. Ejercicios Propuestos

Ejercicio 1: Sistema de Mantenimiento

1. Define una interfaz `Mantenimiento` que incluya métodos para reparaciones y limpieza.
2. Refactoriza el sistema para separar estos métodos en interfaces específicas para cada tipo de tarea.

Ejercicio 2: Sistema de Operaciones Bancarias

1. Define una interfaz `OperacionBancaria` que incluya métodos para transferencias, retiros, y pagos de facturas.
2. Divide la interfaz en varias interfaces más específicas y refactoriza las clases que las implementan.

Ejercicio 3: Gestión de Vehículos

1. Define una interfaz `Vehiculo` que incluya métodos para conducir y cargar mercancías.
2. Refactoriza el sistema para tener interfaces más específicas para la conducción y el transporte de mercancías.

Conclusión del Taller:

Este taller ha proporcionado una comprensión profunda del **Principio de Segregación de Interfaces (ISP)** y cómo aplicarlo para crear interfaces más específicas que se adapten mejor a las necesidades de las clases que las implementan. Al aplicar el ISP, los estudiantes aprenderán a reducir el acoplamiento y aumentar la cohesión en sus sistemas orientados a objetos.