Programación orientada a objetos Individual - POO - Unidad 2
Actividad: Taller sobre Interfaces vs Clases Abstractas en Java
Tutor: JOHN CARLOS ARRIETA ARRIETA DOCENTE

Taller: Interfaces vs Clases Abstractas en Java

Objetivo del Taller:

En este taller, los estudiantes aprenderán las diferencias clave entre interfaces y clases abstractas en Java, cuándo usar cada una, y cómo se pueden complementar. Se proporcionarán explicaciones claras sobre los propósitos de cada una, su sintaxis, y cómo afectan la estructura de un programa orientado a objetos. Además, se proporcionarán ejemplos prácticos y ejercicios para consolidar los conceptos.

Temario:

1. **Contextualización y Definición de Interfaces y Clases Abstractas**
   - ¿Qué son las interfaces y las clases abstractas?
   - Propósitos y diferencias fundamentales.
2. **Objetivos de las Interfaces y Clases Abstractas**
   - Definir contratos y comportamientos comunes.
   - Desacoplar la implementación y la definición.
3. **Cuándo Usar y Cuándo No Usar Interfaces y Clases Abstractas**
   - Casos recomendados para cada una.
   - Cuándo usar una combinación de ambas.
4. **Sintaxis y Estructura de Interfaces y Clases Abstractas**
   - Definición de métodos abstractos y concretos.
   - Métodos predeterminados y estáticos en interfaces.
5. **Ejemplos de Uso Correcto**
   - Ejemplos prácticos de implementación de interfaces y clases abstractas.
6. **Ejemplos de Uso Incorrecto con Errores de Compilación**
   - Ejemplos que violan las reglas y no compilan.
7. **Ejemplos de Uso Incorrecto sin Generar Error de Compilación**
   - Ejemplos que son malas prácticas pero que no generan errores.
8. **Ejercicios Propuestos**
   - Ejercicios para poner en práctica el uso correcto de interfaces y clases abstractas.

---

1. Contextualización y Definición de Interfaces y Clases Abstractas

¿Qué son las Interfaces y las Clases Abstractas?

En Java, tanto las interfaces como las clases abstractas se utilizan para definir comportamientos comunes y crear una base que otras clases deben implementar o extender. Sin embargo, existen diferencias clave entre ambas.

- **Interfaces:** Son colecciones de métodos abstractos (sin cuerpo) que definen un contrato que las clases deben implementar. A partir de Java 8, las interfaces pueden tener métodos predeterminados (`default`) y métodos estáticos.
- **Clases Abstractas:** Son clases que no se pueden instanciar y que pueden tener métodos abstractos y métodos concretos. Se utilizan para definir una base común para una jerarquía de clases.

---

2. Objetivos de las Interfaces y Clases Abstractas

Objetivos Clave:

1. **Definir Contratos:** Las interfaces definen un conjunto de métodos que todas las clases que las implementen deben cumplir, garantizando consistencia y limpieza en el código.
2. **Desacoplar la Implementación:** Al separar la definición del comportamiento de su implementación específica, las interfaces facilitan un diseño flexible y desacoplado.
3. **Crear una Base Común:** Las clases abstractas proporcionan una base común que permite la reutilización de código y la definición de comportamientos comunes en una jerarquía de clases.

---

3. Cuándo Usar y Cuándo No Usar Interfaces y Clases Abstractas

Cuándo Usar Interfaces:

- Cuando se necesita definir un conjunto de métodos que diferentes clases deben implementar.
- Cuando se desea proporcionar un contrato de comportamiento para clases no relacionadas.
- Cuando se requiere definir capacidades sin tener que proporcionar una implementación común.

Cuándo Usar Clases Abstractas:

- Cuando se necesita crear una clase base común que otras clases deben extender.
- Cuando se desea proporcionar una implementación común para algunas clases relacionadas.
- Cuando se necesita definir tanto métodos abstractos como métodos concretos que serán compartidos.

Cuándo Usar Ambos:

- Cuando se desea definir un contrato de comportamiento con una interfaz y, al mismo tiempo, proporcionar una base común para clases relacionadas con una clase abstracta.
- Cuando se requiere flexibilidad y una estructura lógica en la jerarquía de clases.

---

4. Sintaxis y Estructura de Interfaces y Clases Abstractas

Definición de Métodos Abstractos en Clases Abstractas:

```
// Clase abstracta
public abstract class Empleado {
    protected String nombre;

    public Empleado(String nombre) {
        this.nombre = nombre;
    }

    public abstract void calcularSalario(); // Método abstracto

    public void mostrarNombre() {
        System.out.println("Nombre: " + nombre);
    }
}
```

Definición de Métodos Predeterminados y Métodos Abstractos en Interfaces:

```
// Definición de una interfaz
public interface Trabajador {
    void trabajar(); // Método abstracto

    default void tomarDescanso() {
        System.out.println("Tomando un descanso.");
    }
}
```

Implementación de Múltiples Interfaces:

```
public class Persona implements Trabajador, Hablador {
    @Override
    public void trabajar() {
        System.out.println("Persona trabajando.");
    }

    @Override
    public void hablar() {
        System.out.println("Persona hablando.");
    }
}
```

Reglas Básicas:

- Una clase abstracta puede tener métodos abstractos y métodos concretos.
- Una interfaz puede tener métodos abstractos, métodos predeterminados y métodos estáticos.
- Las clases pueden implementar múltiples interfaces pero solo pueden extender una clase abstracta.

---

5. Ejemplos de Uso Correcto

Ejemplo Correcto 1: Uso de una Clase Abstracta como Base Común

```java
// Clase abstracta
public abstract class Figura {
    public abstract double calcularArea(); // Método abstracto

    public void mostrarFigura() {
        System.out.println("Mostrando figura.");
    }
}

// Clase derivada
public class Circulo extends Figura {
    private double radio;

    public Circulo(double radio) {
        this.radio = radio;
    }

    @Override
    public double calcularArea() {
        return Math.PI * radio * radio;
    }
}
```

Ejemplo Correcto 2: Uso de Interfaces para Definir Contratos

```java
// Definición de una interfaz
public interface Vehiculo {
    void acelerar();

    default void encender() {
        System.out.println("El vehículo está encendido.");
    }
}

// Clase que implementa la interfaz
public class Coche implements Vehiculo {
    @Override
    public void acelerar() {
        System.out.println("El coche está acelerando.");
    }
}
```

---

6. Ejemplos de Uso Incorrecto con Errores de Compilación

Ejemplo Incorrecto 1: No Implementar Todos los Métodos de la Interfaz

```java
// Definición de una interfaz
public interface Jugador {
    void jugar();
}

// Clase incorrecta que no implementa el método jugar
public class Futbolista implements Jugador {
    // Error de compilación: La clase Futbolista debe implementar el método jugar()
}
```

Explicación: La clase `Futbolista` no proporciona una implementación para el método `jugar`, lo cual genera un error de compilación.

---

7. Ejemplos de Uso Incorrecto sin Generar Error de Compilación

Ejemplo Incorrecto 2: Definir Métodos Innecesarios en una Clase Abstracta

```
// Clase abstracta
public abstract class Animal {
    public abstract void hacerSonido();
}

// Clase derivada incorrecta
public class Gato extends Animal {
    // Aunque el método hacerSonido no se sobrescribe, el error se evita por falta de visibilidad o error lógico.
}
```

Explicación: Aunque el código compila, esta es una mala práctica ya que se espera que todas las clases derivadas proporcionen una implementación de los métodos abstractos definidos en la clase base.

---

8. Ejercicios Propuestos

Ejercicio 1: Clase `Figura` con Clases Derivadas `Rectangulo` y `Triangulo`

1. Define una clase abstracta `Figura` con un método abstracto `calcularArea`.
2. Crea dos clases derivadas (`Rectangulo` y `Triangulo`) que extiendan `Figura` y proporcionen sus propias implementaciones de `calcularArea`.
3. Crea una clase de prueba que instancie objetos de `Rectangulo` y `Triangulo` y muestre sus áreas.

Ejercicio 2: Interfaces `Volador` y `Nadador`

1. Define una interfaz `Volador` con un método `volar`.
2. Define una interfaz `Nadador` con un método `nadar`.
3. Crea una clase `Pato` que implemente ambas interfaces y defina sus métodos.
4. Crea una clase de prueba para instanciar un objeto de `Pato` y usar ambos métodos.

Ejercicio 3: Uso Incorrecto de Clases Abstractas e Interfaces

1. Intenta crear una clase concreta que implemente una interfaz sin proporcionar implementaciones de sus métodos y observa los errores de compilación.
2. Intenta definir métodos predeterminados en una clase abstracta y discute por qué esto no es posible.

Conclusión del Taller:

En este taller, los estudiantes han aprendido las diferencias clave entre interfaces y clases abstractas, cuándo usar cada una, y cómo combinarlas para crear una arquitectura más flexible y lógica. Ambos conceptos son fundamentales para diseñar software de manera modular y escalable.

Puntos Clave a Recordar:

- Las interfaces definen contratos de comportamiento que las clases deben cumplir.
- Las clases abstractas proporcionan una base común para una jerarquía de clases.
- Las interfaces permiten la herencia múltiple, mientras que las clases abstractas no.
- A partir de Java 8, las interfaces pueden tener métodos predeterminados y estáticos.