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

Taller: Clases Abstractas en Java

Objetivo del Taller:

En este taller, los estudiantes aprenderán qué son las clases abstractas en Java, cuándo y cómo utilizarlas correctamente. Se explicará cómo se definen, qué tipo de métodos pueden tener, y las diferencias fundamentales entre una clase abstracta y una clase concreta o una interfaz. Además, se proporcionarán ejemplos prácticos y ejercicios para reforzar estos conceptos y evitar errores comunes.

Temario:

1. **Contextualización y Definición de Clases Abstractas**
   - ¿Qué es una clase abstracta?
   - Diferencias entre clases abstractas, interfaces y clases concretas.
2. **Objetivos del Uso de Clases Abstractas**
   - Reutilización de código y definición de comportamientos comunes.
   - Crear una base común para una jerarquía de clases.
3. **Cuándo Usar y Cuándo No Usar Clases Abstractas**
   - Casos recomendados y no recomendados.
4. **Sintaxis de las Clases Abstractas**
   - Definición de métodos abstractos y métodos concretos en clases abstractas.
   - Herencia de clases abstractas.
5. **Ejemplos de Uso Correcto**
   - Ejemplos prácticos que muestren la definición y uso de clases abstractas.
6. **Ejemplos de Uso Incorrecto con Errores de Compilación**
   - Ejemplos que violan las reglas de las clases abstractas 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 e incorrecto de clases abstractas.

---

1. Contextualización y Definición de Clases Abstractas

¿Qué es una Clase Abstracta?

Una clase abstracta en Java es una clase que no se puede instanciar y se utiliza para definir una base común para otras clases. Las clases abstractas pueden contener métodos abstractos (sin cuerpo) que deben ser implementados por las clases derivadas, así como métodos concretos (con cuerpo) que proporcionan una implementación común.

Diferencias Entre Clases Abstractas, Interfaces y Clases Concretas:

- **Clases Abstractas**: Pueden tener métodos abstractos y concretos. También pueden tener atributos y constructores.
- **Interfaces**: Solo pueden tener métodos abstractos (a menos que se definan como predeterminados o estáticos). No pueden tener constructores.
- **Clases Concretas**: Pueden tener métodos concretos y atributos, y pueden ser instanciadas directamente.

---

2. Objetivos del Uso de Clases Abstractas

Objetivos Clave:

1. **Reutilización de Código**: Las clases abstractas permiten definir métodos comunes para una jerarquía de clases, evitando la duplicación de código.
2. **Definir Comportamientos Comunes**: Proporcionan una estructura base que las clases derivadas pueden extender y especializar.
3. **Crear una Jerarquía de Clases**: Ayudan a organizar y estructurar el código en una jerarquía lógica.

---

3. Cuándo Usar y Cuándo No Usar Clases Abstractas

Cuándo Usar Clases Abstractas:

- Cuando se necesita crear una clase base común que otras clases deben extender.
- Cuando se desea definir un comportamiento común en la clase base y dejar que las clases derivadas proporcionen sus propias implementaciones de ciertos métodos.
- Cuando se necesita definir métodos concretos que puedan ser utilizados por todas las clases derivadas.

Cuándo No Usar Clases Abstractas:

- Cuando se necesita definir métodos sin una implementación predeterminada (en este caso, es preferible usar interfaces).
- Cuando se necesita implementar múltiples comportamientos no relacionados.

- Cuando no existe una relación lógica de "es-un" entre la clase base y las clases derivadas.

---

4. Sintaxis de las Clases Abstractas

Definición de Métodos Abstractos y Métodos Concretos en Clases Abstractas:

```
// Definición de una clase abstracta
public abstract class Animal {
    protected String nombre;

    // Constructor
    public Animal(String nombre) {
        this.nombre = nombre;
    }

    // Método abstracto (sin cuerpo)
    public abstract void hacerSonido();

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

// Definición de una clase derivada
public class Perro extends Animal {
    public Perro(String nombre) {
        super(nombre);
    }

    @Override
    public void hacerSonido() {
        System.out.println("El perro ladra.");
    }
}
```

Reglas Básicas:

- Las clases abstractas no pueden ser instanciadas directamente.
- Las clases derivadas deben implementar todos los métodos abstractos definidos en la clase abstracta.
- Se puede utilizar la palabra clave `abstract` tanto en la clase como en los métodos.

---

5. Ejemplos de Uso Correcto

Ejemplo Correcto 1: Clase `Empleado` y Clase `Gerente`

```
// 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);
    }
}
```

```java
// Clase derivada
public class Gerente extends Empleado {
    private double bono;

    public Gerente(String nombre, double bono) {
        super(nombre);
        this.bono = bono;
    }

    @Override
    public void calcularSalario() {
        System.out.println("El salario del gerente incluye un bono de: " + bono);
    }
}
```

Explicación: La clase abstracta `Empleado` define un método abstracto `calcularSalario` y un método concreto `mostrarNombre`. La clase `Gerente` extiende `Empleado` y proporciona su propia implementación del método abstracto.

---

6. Ejemplos de Uso Incorrecto con Errores de Compilación

Ejemplo Incorrecto 1: No Implementar Métodos Abstractos

```java
// Clase abstracta
public abstract class Vehiculo {
    public abstract void conducir(); // Método abstracto
}

// Clase derivada incorrecta
public class Coche extends Vehiculo {
    // Error de compilación: La clase Coche debe implementar el método abstracto conducir()
}
```

Explicación: La clase `Coche` no proporciona una implementación del método abstracto `conducir`, lo cual genera un error de compilación.

---

7. Ejemplos de Uso Incorrecto sin Generar Error de Compilación

Ejemplo Incorrecto 2: Definir Métodos Abstractos que No Son Sobrescritos

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

// Clase derivada que no sobrescribe el método abstracto
public class Gato extends Animal {
    // Aunque el método hacerSonido no es sobrescrito, se permite debido a la 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` y Clases Derivadas `Circulo` y `Rectangulo`

1. Define una clase abstracta `Figura` con un método abstracto `calcularArea`.
2. Crea dos clases derivadas (`Circulo` y `Rectangulo`) que extiendan `Figura` y proporcionen sus propias implementaciones de `calcularArea`.

3. Define un método `mostrarArea` en `Figura` y utiliza `calcularArea` para mostrar el área.

Ejercicio 2: Clase `Empleado` y Clases Derivadas `Gerente` y `Vendedor`

1. Define una clase abstracta `Empleado` con un método abstracto `calcularSalario`.
2. Crea dos clases derivadas (`Gerente` y `Vendedor`) que extiendan `Empleado` y proporcionen sus propias implementaciones de `calcularSalario`.
3. Define un método `mostrarDetalles` en `Empleado` y utiliza `calcularSalario` para mostrar los detalles del empleado.

Ejercicio 3: Uso Incorrecto de Clases Abstractas

1. Intenta instanciar una clase abstracta directamente y observa los errores de compilación.
2. Intenta definir un método concreto en una clase abstracta que sea necesario sobrescribir en las clases derivadas.

Conclusión del Taller:

Este taller ha proporcionado una comprensión profunda sobre las clases abstractas en Java, explicando su propósito, sintaxis y uso correcto. Las clases abstractas son fundamentales para definir una estructura común en una jerarquía de clases y para proporcionar una base sólida para extender el código.

Puntos Clave a Recordar:

- Las clases abstractas no pueden ser instanciadas directamente.
- Una clase abstracta puede tener métodos abstractos y concretos.
- Las clases derivadas deben implementar todos los métodos abstractos de la clase base.
- Las clases abstractas proporcionan una base común y ayudan a evitar la duplicación de código en una jerarquía de clases.