Programación orientada a objetos Individual - POO - Unidad 2
Actividad: Taller sobre el modificador de acceso **private** en Java
Tutor: JOHN CARLOS ARRIETA ARRIETA DOCENTE

Taller: Modificador de Acceso `private` en Java

Objetivo del Taller:

En este taller, los estudiantes aprenderán el uso correcto y la importancia del modificador de acceso `private` en Java. Se proporcionarán definiciones claras, se explicará cuándo y cómo utilizar `private`, y se ofrecerán ejercicios prácticos para reforzar los conceptos.

Temario:

---

1. Contextualización y Definición del Modificador `private`

El modificador `private` en Java se utiliza para restringir el acceso a miembros (atributos y métodos) de una clase. Los miembros marcados como `private` solo pueden ser accedidos y modificados dentro de la misma clase, lo cual permite proteger los datos de modificaciones no controladas o de accesos indebidos.

Definición Técnica:

- Un atributo o método declarado como `private` solo puede ser accedido dentro de la misma clase. No se puede acceder desde otras clases, incluso si pertenecen al mismo paquete.

2. Objetivos del Uso de `private`

El uso del modificador `private` cumple con varios objetivos clave en la programación orientada a objetos:

- **Protección de Datos:** Se asegura de que las propiedades o métodos sensibles solo sean modificados o accedidos a través de métodos controlados.
- **Encapsulamiento:** Permite ocultar los detalles internos de la implementación, exponiendo únicamente una interfaz pública segura (usando métodos `get` y `set`).
- **Control de Acceso:** Restringe el acceso a ciertos miembros, asegurando que las operaciones que se realicen sobre los datos sean seguras y coherentes.

Ejemplo de Encapsulamiento con `private`

```
public class CuentaBancaria {
    private double saldo; // Atributo privado para proteger el saldo

    public CuentaBancaria(double saldoInicial) {
        if (saldoInicial >= 0) {
            this.saldo = saldoInicial;
        } else {
            this.saldo = 0;
        }
    }

    public double getSaldo() {
        return saldo;
    }

    public void depositar(double monto) {
```

```
        if (monto > 0) {
            saldo += monto;
        }
    }
}
```

Explicación: Aquí, `saldo` es un atributo privado y no puede ser accedido directamente desde fuera de la clase. Solo se puede consultar o modificar a través de los métodos públicos `getSaldo` y `depositar`.

3. Cuándo Usar y Cuándo No Usar `private`

Cuándo Usar `private`:

- Cuando se desea proteger un atributo de modificaciones directas.
- Al ocultar la lógica interna de un método que no necesita ser visible desde fuera de la clase.
- Para cumplir con el principio de encapsulamiento y proteger la integridad de los datos.

Cuándo No Usar `private`:

- No se debe usar en métodos o propiedades que se espera que sean accedidos directamente desde otras clases o subclases.
- Evitar usar `private` si se planea proporcionar extensiones o derivaciones de una clase en otras partes del sistema.

4. Ejemplos de Uso Correcto

Ejemplo Correcto 1:

```
public class Persona {
    private String nombre;
    private int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        if (edad >= 0) {
            this.edad = edad;
        }
    }
}
```

Explicación: Los atributos `nombre` y `edad` son privados y solo se puede acceder a ellos mediante métodos `get` y `set`. Además, se incluyen validaciones para asegurar la coherencia de los datos.

Ejemplo Correcto 2:

```
public class Vehiculo {
    private String marca;
    private double velocidad;

    public Vehiculo(String marca, double velocidad) {
        this.marca = marca;
        this.velocidad = velocidad;
    }
```

```
    public String getMarca() {
        return marca;
    }

    public double getVelocidad() {
        return velocidad;
    }

    public void acelerar(double incremento) {
        if (incremento > 0) {
            velocidad += incremento;
        }
    }
}
```

Explicación: El atributo `velocidad` es privado y solo puede ser modificado utilizando el método `acelerar`, que incluye una validación.

5. Ejemplos de Uso Incorrecto con Errores de Compilación

Ejemplo Incorrecto 1: Intentar acceder a un miembro privado desde otra clase

```
public class Persona {
    private String nombre;

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

public class Main {
    public static void main(String[] args) {
        Persona p = new Persona("Juan");
        System.out.println(p.nombre); // Error de compilación: nombre tiene acceso privado
    }
}
```

Explicación: Intentar acceder a `nombre` directamente desde la clase `Main` genera un error de compilación porque `nombre` tiene un modificador de acceso `private`.

6. Ejemplos de Uso Incorrecto sin Generar Error de Compilación

Ejemplo Incorrecto 2: No usar métodos `get` y `set` para controlar el acceso

```
public class Producto {
    private double precio;

    public Producto(double precio) {
        this.precio = precio;
    }

    public void actualizarPrecio(double nuevoPrecio) {
        // No se realiza ninguna validación aquí
        precio = nuevoPrecio;
    }
}
```

Explicación: Aunque el atributo `precio` es privado, la falta de validaciones adecuadas en el método `actualizarPrecio` permite asignar valores negativos o incorrectos, lo cual es una mala práctica.

7. Ejercicios Propuestos

Ejercicio 1: Clase `Estudiante`

Crea una clase `Estudiante` con las propiedades `nombre`, `edad` y `notaPromedio`. Define:

1. Todas las propiedades deben ser privadas.
2. Proporciona métodos `get` y `set` para cada propiedad, incluyendo validaciones en los métodos `set`.
3. Crea una clase de prueba para crear instancias de `Estudiante` y acceder a las propiedades.

Ejercicio 2: Clase `Coche`

Crea una clase `Coche` con las propiedades `marca`, `modelo` y `velocidadMaxima`. Define:

1. Todas las propiedades deben ser privadas.
2. Proporciona un método `acelerar` que incremente `velocidadMaxima` solo si el incremento es positivo.
3. Crea una clase de prueba que intente acceder a las propiedades directamente y observa los errores de compilación.

Ejercicio 3: Uso Incorrecto de `private`

1. Intenta crear una clase que declare una propiedad `private` y accede a ella directamente desde otra clase. Observa los errores de compilación.
2. Crea una clase que no implemente métodos `get` y `set` para un atributo `private`, y discute cómo podría afectar la protección de datos.

Conclusión del Taller:

Este taller ha proporcionado una base sólida sobre el modificador de acceso `private`, explicando su propósito, cuándo usarlo, y cómo implementar correctamente el encapsulamiento para proteger los datos y controlar el acceso a los mismos. Además, se han discutido errores comunes y se han ofrecido ejemplos prácticos para reforzar el aprendizaje.