Principios de Encapsulamiento

El encapsulamiento es un principio fundamental de la programación orientada a objetos (POO) que consiste en restringir el acceso a los detalles internos de una clase y exponer sólo lo que es necesario para su uso externo. Esto se logra mediante modificadores de acceso y el uso de métodos específicos para acceder y modificar los atributos privados de una clase.

Modificadores de Acceso

En Java, existen tres modificadores de acceso principales:

1. **public**: El miembro (atributo o método) es accesible desde cualquier otra clase.
2. **private**: El miembro es accesible solo dentro de la clase donde está declarado.
3. **protected**: El miembro es accesible dentro de su propio paquete y por las subclases.

Uso de `this`

La palabra clave `this` se utiliza dentro de un método o constructor para referirse al objeto actual. Se utiliza comúnmente para diferenciar entre los atributos de una clase y los parámetros de un método o constructor cuando tienen el mismo nombre.

Ejercicios Prácticos

Ejercicio 1: Clase Persona con Encapsulamiento

Crea una clase `Persona` con los atributos privados `nombre` y `edad`. Proporciona métodos públicos para acceder y modificar estos atributos.

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

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

    // Getter para nombre
    public String getNombre() {
        return nombre;
    }

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

    // Getter para edad
```

```
    public int getEdad() {
        return edad;
    }

    // Setter para edad
    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

Ejercicio 2: Clase CuentaBancaria

Crea una clase `CuentaBancaria` con los atributos privados `numeroCuenta` y `saldo`. Proporciona métodos para depositar y retirar dinero, asegurando que el saldo nunca sea negativo.

```
public class CuentaBancaria {
    private String numeroCuenta;
    private double saldo;

    public CuentaBancaria(String numeroCuenta, double saldoInicial) {
        this.numeroCuenta = numeroCuenta;
        this.saldo = saldoInicial;
    }

    public String getNumeroCuenta() {
        return numeroCuenta;
    }

    public double getSaldo() {
        return saldo;
    }

    public void depositar(double monto) {
        if (monto > 0) {
            this.saldo += monto;
        }
    }

    public void retirar(double monto) {
        if (monto > 0 && monto <= saldo) {
            this.saldo -= monto;
        }
    }
}
```

Ejercicio 3: Clase Producto

Implementa una clase `Producto` con los atributos privados `codigo`, `nombre`, y `precio`. Crea métodos para establecer y obtener el valor de cada atributo.

```
public class Producto {
    private String codigo;
    private String nombre;
    private double precio;

    public Producto(String codigo, String nombre, double precio) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.precio = precio;
    }

    public String getCodigo() {
        return codigo;
    }

    public void setCodigo(String codigo) {
        this.codigo = codigo;
    }

    public String getNombre() {
        return nombre;
    }

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

    public double getPrecio() {
        return precio;
    }

    public void setPrecio(double precio) {
        this.precio = precio;
    }
}
```

Ejercicio 4: Clase Estudiante

Crea una clase `Estudiante` que tenga los atributos privados `matricula`, `nombre`, y `notaFinal`. Añade métodos para establecer y obtener cada atributo.

```
public class Estudiante {
    private String matricula;
    private String nombre;
    private double notaFinal;

    public Estudiante(String matricula, String nombre, double notaFinal) {
        this.matricula = matricula;
        this.nombre = nombre;
        this.notaFinal = notaFinal;
```

```
    }

    public String getMatricula() {
        return matricula;
    }

    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }

    public String getNombre() {
        return nombre;
    }

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

    public double getNotaFinal() {
        return notaFinal;
    }

    public void setNotaFinal(double notaFinal) {
        this.notaFinal = notaFinal;
    }
}
```

Ejercicio 5: Clase Empleado

Crea una clase `Empleado` con los atributos privados `id`, `nombre`, y `salario`. Proporciona métodos para acceder y modificar estos atributos.

```
public class Empleado {
    private int id;
    private String nombre;
    private double salario;

    public Empleado(int id, String nombre, double salario) {
        this.id = id;
        this.nombre = nombre;
        this.salario = salario;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

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

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

    public double getSalario() {
        return salario;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }
}
```

Ejercicio 6: Clase Vehiculo

Implementa una clase `Vehiculo` con atributos privados `marca`, `modelo`, y `año`. Proporciona métodos para acceder y modificar estos atributos.

```
public class Vehiculo {
    private String marca;
    private String modelo;
    private int año;

    public Vehiculo(String marca, String modelo, int año) {
        this.marca = marca;
        this.modelo = modelo;
        this.año = año;
    }

    public String getMarca() {
        return marca;
    }

    public void setMarca(String marca) {
        this.marca = marca;
    }

    public String getModelo() {
        return modelo;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }

    public int getAño() {
        return año;
```

```
    }

    public void setAño(int año) {
        this.año = año;
    }
}
```

Ejercicio 7: Clase Libro

Crea una clase `Libro` con los atributos privados `titulo`, `autor`, y `isbn`. Implementa métodos para establecer y obtener el valor de estos atributos.

```
public class Libro {
    private String titulo;
    private String autor;
    private String isbn;

    public Libro(String titulo, String autor, String isbn) {
        this.titulo = titulo;
        this.autor = autor;
        this.isbn = isbn;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getAutor() {
        return autor;
    }

    public void setAutor(String autor) {
        this.autor = autor;
    }

    public String getIsbn() {
        return isbn;
    }

    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }
}
```

Ejercicio 8: Clase Animal

Crea una clase `Animal` con los atributos privados `especie`, `nombre`, y `edad`. Proporciona métodos para acceder y modificar estos atributos.

```
public class Animal {
    private String especie;
    private String nombre;
    private int edad;

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

    public String getEspecie() {
        return especie;
    }

    public void setEspecie(String especie) {
        this.especie = especie;
    }

    public String getNombre() {
        return nombre;
    }

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

    public int getEdad() {
        return edad;
    }

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

Ejercicio 9: Clase Tienda

Implementa una clase `Tienda` con los atributos privados `nombre`, `direccion`, y `productos` (una lista de objetos `Producto`). Proporciona métodos para agregar y eliminar productos, así como para listar los productos disponibles.

```
import java.util.ArrayList;
import java.util.List;

public class Tienda {
    private String nombre;
```

```
    private String direccion;
    private List<Producto> productos;

    public Tienda(String nombre, String direccion) {
        this.nombre = nombre;
        this.direccion = direccion;
        this.productos = new ArrayList<>();
    }

    public String getNombre() {
        return nombre;
    }

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

    public String getDireccion() {
        return direccion;
    }

    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }

    public void agregarProducto(Producto producto) {
        productos.add(producto);
    }

    public void eliminarProducto(Producto producto) {
        productos.remove(producto);
    }

    public List<Producto> getProductos() {
        return productos;
    }
}
```

Ejercicio 10: Clase Hospital

Crea una clase `Hospital` con los atributos privados `nombre`, `direccion`, y `pacientes` (una lista de objetos `Persona`). Proporciona métodos para agregar y eliminar pacientes, así como para listar los pacientes registrados.

```
import java.util.ArrayList;
import java.util.List;

public class Hospital {
    private String nombre;
    private String direccion;
    private List<Persona> pacientes;
```

```
    public Hospital(String nombre, String direccion) {
        this.nombre = nombre;
        this.direccion = direccion;
        this.pacientes = new ArrayList<>();
    }

    public String getNombre() {
        return nombre;
    }

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

    public String getDireccion() {
        return direccion;
    }

    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }

    public void agregarPaciente(Persona paciente) {
        pacientes.add(paciente);
    }

    public void eliminarPaciente(Persona paciente) {
        pacientes.remove(paciente);
    }

    public List<Persona> getPacientes() {
        return pacientes;
    }
}
```

Ejemplos de Uso de Clases con Encapsulamiento en Java

A continuación, se presentan algunos ejemplos de cómo utilizar las clases con encapsulamiento que hemos creado en los ejercicios anteriores.

Ejemplo 1: Uso de la Clase `Persona`

```
public class Main {
    public static void main(String[] args) {
        Persona persona = new Persona("Juan Perez", 30);

        // Accediendo y modificando atributos usando métodos getter y setter
```

```
        System.out.println("Nombre: " + persona.getNombre());
        System.out.println("Edad: " + persona.getEdad());

        persona.setNombre("Carlos Gomez");
        persona.setEdad(35);

        System.out.println("Nuevo Nombre: " + persona.getNombre());
        System.out.println("Nueva Edad: " + persona.getEdad());
    }
}
```

Ejemplo 2: Uso de la Clase `CuentaBancaria`

```
public class Main {
    public static void main(String[] args) {
        CuentaBancaria cuenta = new CuentaBancaria("123456789", 500.0);

        // Depósito y retiro
        cuenta.depositar(200.0);
        System.out.println("Saldo después del depósito: " + cuenta.getSaldo());

        cuenta.retirar(100.0);
        System.out.println("Saldo después del retiro: " + cuenta.getSaldo());

        cuenta.retirar(700.0); // No debe permitir el retiro
        System.out.println("Saldo después de intento de retiro excesivo: " +
cuenta.getSaldo());
    }
}
```

Ejemplo 3: Uso de la Clase `Producto`

```
public class Main {
    public static void main(String[] args) {
        Producto producto = new Producto("P001", "Laptop", 1500.0);

        // Accediendo y modificando atributos
        System.out.println("Código: " + producto.getCodigo());
        System.out.println("Nombre: " + producto.getNombre());
        System.out.println("Precio: " + producto.getPrecio());

        producto.setNombre("Laptop Gaming");
        producto.setPrecio(2000.0);

        System.out.println("Nuevo Nombre: " + producto.getNombre());
        System.out.println("Nuevo Precio: " + producto.getPrecio());
    }
}
```

Ejemplo 4: Uso de la Clase `Estudiante`

```
public class Main {
    public static void main(String[] args) {
        Estudiante estudiante = new Estudiante("202301", "Ana Lopez", 9.5);

        // Accediendo y modificando atributos
        System.out.println("Matrícula: " + estudiante.getMatricula());
        System.out.println("Nombre: " + estudiante.getNombre());
        System.out.println("Nota Final: " + estudiante.getNotaFinal());

        estudiante.setNotaFinal(10.0);

        System.out.println("Nueva Nota Final: " + estudiante.getNotaFinal());
    }
}
```

Ejemplo 5: Uso de la Clase `Empleado`

```
public class Main {
    public static void main(String[] args) {
        Empleado empleado = new Empleado(1, "Luis Fernandez", 3000.0);

        // Accediendo y modificando atributos
        System.out.println("ID: " + empleado.getId());
        System.out.println("Nombre: " + empleado.getNombre());
        System.out.println("Salario: " + empleado.getSalario());

        empleado.setSalario(3500.0);

        System.out.println("Nuevo Salario: " + empleado.getSalario());
    }
}
```

Ejemplo 6: Uso de la Clase `Vehiculo`

```
public class Main {
    public static void main(String[] args) {
        Vehiculo vehiculo = new Vehiculo("Toyota", "Corolla", 2020);

        // Accediendo y modificando atributos
        System.out.println("Marca: " + vehiculo.getMarca());
        System.out.println("Modelo: " + vehiculo.getModelo());
        System.out.println("Año: " + vehiculo.getAño());

        vehiculo.setAño(2021);
```

```
        System.out.println("Nuevo Año: " + vehiculo.getAño());
    }
}
```

Ejemplo 7: Uso de la Clase `Libro`

```
public class Main {
    public static void main(String[] args) {
        Libro libro = new Libro("1984", "George Orwell", "978-0451524935");

        // Accediendo y modificando atributos
        System.out.println("Título: " + libro.getTitulo());
        System.out.println("Autor: " + libro.getAutor());
        System.out.println("ISBN: " + libro.getIsbn());

        libro.setTitulo("Animal Farm");
        libro.setIsbn("978-0451526342");

        System.out.println("Nuevo Título: " + libro.getTitulo());
        System.out.println("Nuevo ISBN: " + libro.getIsbn());
    }
}
```

Ejemplo 8: Uso de la Clase `Animal`

```
public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal("Canis", "Firulais", 5);

        // Accediendo y modificando atributos
        System.out.println("Especie: " + animal.getEspecie());
        System.out.println("Nombre: " + animal.getNombre());
        System.out.println("Edad: " + animal.getEdad());

        animal.setEdad(6);

        System.out.println("Nueva Edad: " + animal.getEdad());
    }
}
```

Ejemplo 9: Uso de la Clase `Tienda`

```
public class Main {
    public static void main(String[] args) {
        Tienda tienda = new Tienda("Tech Store", "Calle Falsa 123");
```

```
        Producto p1 = new Producto("P001", "Laptop", 1500.0);
        Producto p2 = new Producto("P002", "Smartphone", 800.0);

        tienda.agregarProducto(p1);
        tienda.agregarProducto(p2);

        System.out.println("Productos en la tienda:");
        for (Producto producto : tienda.getProductos()) {
            System.out.println(producto.getNombre() + " - " +
producto.getPrecio());
        }
    }
}
```

Ejemplo 10: Uso de la Clase `Hospital`

```
public class Main {
    public static void main(String[] args) {
        Hospital hospital = new Hospital("Hospital General", "Av. Salud 456");

        Persona paciente1 = new Persona("Maria Gomez", 45);
        Persona paciente2 = new Persona("Pedro Martinez", 60);

        hospital.agregarPaciente(paciente1);
        hospital.agregarPaciente(paciente2);

        System.out.println("Pacientes registrados en el hospital:");
        for (Persona paciente : hospital.getPacientes()) {
            System.out.println(paciente.getNombre() + " - " + paciente.getEdad() +
" años");
        }
    }
}
```

En Java, el modificador de acceso protected se utiliza para permitir que los miembros de una clase (como campos, métodos o constructores) sean accesibles dentro del mismo paquete y también por las subclases, incluso si estas subclases están en paquetes diferentes.

Aquí tienes un ejemplo que ilustra el uso de protected:

```
// Paquete 1
package paquete1;

public class ClasePadre {
    // Campo protegido
    protected int valorProtegido = 10;

    // Método protegido
```

```
    protected void mostrarValor() {
        System.out.println("El valor protegido es: " + valorProtegido);
    }
}
```

```
// Paquete 2
package paquete2;

import paquete1.ClasePadre;

public class ClaseHija extends ClasePadre {
    public void accederProtegido() {
        // Acceso al campo protegido de la clase padre
        valorProtegido = 20;
        System.out.println("Valor protegido modificado en la clase hija: " +
valorProtegido);

        // Llamada al método protegido de la clase padre
        mostrarValor();
    }
}
```

```
// Clase principal para probar el ejemplo
public class Main {
    public static void main(String[] args) {
        ClaseHija hija = new ClaseHija();
        hija.accederProtegido();
    }
}
```

Explicación

`ClasePadre`: Tiene un campo valorProtegido y un método `mostrarValor()`, ambos marcados como `protected`.

Esto significa que solo las clases dentro del mismo paquete (paquete1) y las subclases (incluso en otros paquetes) pueden acceder a estos miembros.

`ClaseHija`: Extiende `ClasePadre` y está en un paquete diferente (paquete2).

Puede acceder al campo valorProtegido y al método `mostrarValor()` porque son protected y `ClaseHija` es una subclase de `ClasePadre`.

`Main`: Crea una instancia de `ClaseHija` y llama al método `accederProtegido()`, que a su vez accede y modifica el campo protegido y llama al método protegido de la clase padre.

```
Salida esperada:
Valor protegido modificado en la clase hija: 20
El valor protegido es: 20
```

Este ejemplo muestra cómo `protected` permite el acceso a miembros de una clase desde subclases, incluso si están en paquetes diferentes.

En Java, el modificador de acceso `protected` no se puede usar en ciertos contextos o situaciones. Aquí te muestro un caso donde no se puede usar `protected` y por qué:

Caso: Miembros de una clase en un contexto no relacionado con herencia o paquete Supongamos que tienes una clase `ClaseA` con un miembro protected, y otra clase `ClaseB` que no es una subclase de `ClaseA` y no está en el mismo paquete. En este caso, `ClaseB` no podrá acceder al miembro protected de `ClaseA`.

Ejemplo:

```java
// Paquete 1
package paquete1;

public class ClaseA {
    // Miembro protegido
    protected int valorProtegido = 42;
}
```

```java
// Paquete 2
package paquete2;

import paquete1.ClaseA;

public class ClaseB {
    public void intentarAcceder() {
        ClaseA instanciaA = new ClaseA();

        // Intento de acceso al miembro protegido
        // Esto generará un error de compilación
        // System.out.println(instanciaA.valorProtegido); // ERROR
    }
}
```

```java
// Clase principal para probar el ejemplo
public class Main {
    public static void main(String[] args) {
        ClaseB b = new ClaseB();
        b.intentarAcceder();
```

```
    }
}
```

Explicación:

`ClaseA`: Tiene un campo valorProtegido marcado como protected.

Esto significa que solo es accesible desde:

- Clases dentro del mismo paquete (paquete1).
- Subclases de ClaseA, incluso si están en otros paquetes.

`ClaseB`:

- Está en un paquete diferente (paquete2).
- No es una subclase de ClaseA.
- Intenta acceder al campo valorProtegido de una instancia de ClaseA.
- Esto no está permitido porque ClaseB no cumple con las condiciones para acceder a un miembro protected.
- Error de compilación:
- El código en ClaseB generará un error de compilación como:

```
error: valorProtegido has protected access in ClaseA
System.out.println(instanciaA.valorProtegido);
```

¿Por qué no se puede usar protected en este caso? El modificador protected está diseñado para permitir el acceso a miembros de una clase solo desde `subclases` o clases dentro del mismo paquete.

Si una clase no es una `subclase` y no está en el mismo paquete, no puede acceder a miembros protected.

Alternativa: Si necesitas que `ClaseB` acceda a `valorProtegido`, tendrías que:

- Cambiar el modificador de acceso a public (no recomendado si no es necesario).
- Hacer que `ClaseB` sea una subclase de `ClaseA`.
- Mover `ClaseB` al mismo paquete que `ClaseA`.